MASTER EN SCIENCES MATHÉMATIQUES

Une nouvelle méthode de classification automatique symbolique basée sur le processus de Poisson homogène

Vanderpypen, Joël

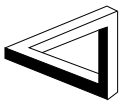
Award date:
2006

FUNDP
NAMUR

Faculté des Sciences
Département de Mathématique

Rempart de la Vierge, 8
B - 5000 Namur (Belgique)

Une nouvelle méthode de classification automatique symbolique basée sur le processus de Poisson homogène



Mémoire présenté pour l'obtention
du grade de
Licencié en Sciences Mathématiques
par

VANDERPYPEN Joël

Promoteur : HARDY André

Année Académique 2005-2006

Résumé : Ce mémoire présente une nouvelle méthode de classification automatique adaptée aux données symboliques. Celle-ci est hiérarchique divisive monothétique, et est basée sur le processus de Poisson homogène. Elle crée un arbre de classification en sélectionnant, à chaque étape, une variable pour couper un groupe en deux sous-ensembles. Le critère utilisé est obtenu par la méthode du maximum de vraisemblance. Il consiste à couper au milieu du plus grand trou entre les données. Une phase dite d'élagage, basée sur le Gap Test, se charge ensuite de simplifier l'arbre. Des exemples (bivariés artificiels puis réels) illustrent le fonctionnement de la nouvelle méthode et la compare à d'autres méthodes déjà existantes.

Summary : This dissertation introduces a new clustering method, which deals with symbolic data. The method is hierarchical divisive monothetic, and is based on the Homogeneous Poisson Process. It builds a clustering tree by selecting, at each step, a variable to cut a group in two subsets. Its criterion is deduced from the criterion of maximum likelihood. It cuts in the middle of the biggest gap between the data. Then a phase based on the Gap Test named pruning has to simplify the tree. Examples (artificial with two variables, then real) show how the new method works, and compares it with other existing clustering methods.

Promoteur (Advisor) : Prof. A. Hardy

Avant toute chose, je tiens à exprimer ma gratitude envers Monsieur André Hardy, promoteur de ce mémoire, pour m'avoir guidé et éclairé tout au long de ce travail. Je voudrais également remercier l'ensemble des professeurs et assistants du département, pour la disponibilité dont ils ont fait preuve envers moi tout au long de mes études, et tout spécialement François Roland.

J'aimerais aussi remercier mes amis de licence, pour les bons moments passés ensemble durant ces quatre années. Que le temps file vite !

Je voudrais également avoir une pensée pour ma famille et mes proches, qui m'ont encouragé au quotidien, et sans qui je n'aurais sans doute pas débuté ces études.

A tous ceux là, et à tous ceux que j'oublie, un grand merci !

Joël

Table des matières

Table des figures

Introduction

Avec l'explosion de l'informatisation et des capacités de stockage, des quantités de données de plus en plus importantes sont collectées et enregistrées. Pensons aux comptabilités ou aux bases de données "client" et "personnel" de grosses entreprises. Le besoin se fait maintenant sentir de pouvoir en extraire des connaissances "cachées", des propriétés qui n'apparaissent pas au premier abord, puisque ces données ont été récoltées dans le but d'un traitement individuel, et non d'une analyse globale. Ce type d'analyse porte le nom de "*Data Mining*", et est très en vogue pour le moment. Elle permet de mieux connaître son entreprise, et d'obtenir des outils d'aide à la décision.

Les principales tâches du *Data Mining* sont la recherche de groupes d'objets semblables, la prédiction de valeurs numériques, l'affectation automatique d'objets dans des classes, la description des caractéristiques d'un ensemble d'objets, la recherche de dépendances entre ces caractéristiques, la détection de valeurs aberrantes, etc.

De nombreuses méthodes d'analyse de données servent donc à ces différentes tâches, et notamment les méthodes de classification. Celles-ci consistent à retrouver dans une population d'objets les différentes classes dans lesquelles ils sont naturellement répartis. La classification doit donc regrouper les objets qui ont des propriétés semblables, en tenant compte des variables qui les caractérisent. De plus, la limite entre deux classes n'est pas toujours bien nette ... La classification traite souvent de grandes quantités de données et par conséquent, cette dimension engendre quelques difficultés au niveau de la place mémoire ainsi que du coût algorithmique.

Un moyen de réduire cette dimension est de traiter les objets par groupe, et non individuellement. Pour cependant conserver la spécificité des objets initiaux, on a introduit de nouveaux types de données, plus complexes que de simples valeurs, appelées *données symboliques*.

Le but de ce présent travail est de mettre au point une nouvelle méthode de classification, traitant des objets symboliques. Dans sa thèse, [Pirçon, 2004] nous a présenté cinq nouvelles méthodes de classification classiques, basées sur les processus de Poisson, et dans le cadre du contrat européen *ASSO*, une de ces méthodes a été adaptée pour pouvoir traiter des données symboliques de type intervalle (*SCLASS*, disponible dans le logiciel [SODAS, 2]). Notre nouvelle méthode sera une adaptation similaire d'une autre de ces cinq méthodes, la plus simple et la plus rapide, qui est basée sur le processus de Poisson homogène.

Une application en *C++* sera développée, et nous permettra de comparer cette nouvelle méthode aux autres méthodes symboliques déjà existantes, et principalement à *SCLASS*.

Nous allons commencer par présenter les différents concepts que nous utiliserons, principalement les notions de classification (Chapitre 1) et de processus de Poisson (Chapitre 2). Nous verrons ensuite les différentes méthodes de [Pirçon, 2004], au chapitre 3. Nous continuerons alors par expliquer en quoi consistent ces données symboliques et nous présenterons quelques méthodes traitant ce type de données (Chapitres 4 et 5). Nous détaillerons ensuite notre nouvelle méthode (Chapitre 6), puis nous la testerons sur différents jeux de données (Chapitre 7). Enfin, nous terminerons par mettre en avant les points forts, ainsi que les points faibles de notre nouvelle méthode.

Une annexe présentera plus en détails le fonctionnement du programme informatique, ainsi que la manière dont nous avons manipulé les jeux de données.

Chapitre 1

La classification automatique

Le but de la classification est de retrouver les groupes "naturels" auxquels appartiennent les différents individus d'un échantillon. Il existe deux types de classification. Dans la première, des groupes d'observations sont déjà identifiés, et le problème consiste à trouver les groupes auxquels appartiennent de nouvelles observations. Ce genre de problème porte le nom de **classification supervisée**, également appelée **analyse discriminante**.

Il se peut que les groupes ne soient pas connus a priori et dans ce cas il s'agit de les établir, de sorte que des individus d'un même groupe soient plus "semblables" entre eux que des individus de groupes différents. On fera alors de la **classification non supervisée**, aussi appelée **classification automatique**, ou **clustering**. C'est ce genre de problème que nous allons traiter par la suite.

Ces groupes qu'on va rechercher sont généralement appelés des **classes**. Leur définition reste assez vague. Intuitivement, on pourrait dire qu'une classe est le regroupement d'observations dont les caractéristiques sont assez proches. On peut aussi considérer qu'une classe est un ensemble d'observations à haute densité, entouré d'une zone à faible densité ([Hartigan, 1975]). Mais cela ne s'applique qu'à des variables continues possédant une densité... Une autre manière de caractériser les classes est de dire qu'elles doivent être clairement séparées et qu'elles doivent contenir des observations qui sont proches les unes des autres. C'est le principe d'isolation externe et de cohésion interne.

1.1 Le problème de classification

Le problème de classification (automatique) décrit la situation suivante. On dispose d'un ensemble $E = \{x_1, ..., x_n\}$ de n individus, sur lesquels on a mesuré p variables $(Y_1, ..., Y_p)$. On peut donc modéliser les données sous forme d'une matrice X $(n \times p)$:

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}$$

avec x_{ij} représentant la valeur pour l'individu i de la j ème variable (Y_j).

On va chercher à répartir ces observations dans un nombre limité de classes. Plus précisément, nous allons rechercher une partition $P = \{C_1, ..., C_k\}$ de E en k classes. Certaines méthodes nécessitent de connaître à l'avance ce k, alors que d'autres n'en ont pas besoin. Nous reviendrons plus en détail sur cet aspect dans la section suivante.

Pour grouper les points, on va utiliser un **critère de classification** qui mesure la qualité de toutes les partitions en k classes :

$$W : \mathcal{P}_k \longrightarrow I\!R \ : \ P \longmapsto W(P, K).$$

La partition optimale sera $P^* = \{C_1^*, ..., C_k^*\}$ avec $W(P^*, k) = \min_{P \in \mathcal{P}_k} W(P, k)$.[1]

Remarquons la nature combinatoire du problème. Notons $S(n, k)$ le nombre total de partitions de n objets en k classes, appelé **nombre de Stirling d'ordre 2**. On a par [Jain and Dubes, 1988] que

$$S(n, k) \ = \ \frac{1}{k!} \sum_{i=1}^{k} C_n^i \, (-1)^{k-i} \, i^n.$$

On obtient ainsi qu'il existe 2 375 101 partitions possibles de 15 individus en 3 classes, et 10^{68} partitions de 100 individus en 5 classes. On voit donc que pour des n et k petits, il est possible de calculer la partition optimale, mais en général, il va falloir utiliser une approche heuristique.

[1] $\mathcal{P}_k$ représente l'ensemble de toutes les partitions en k classes possibles de l'ensemble E de départ

1.2 Méthodes hiérarchiques ou de partitionnement ?

Une grande différence entre les méthodes hiérarchiques et les méthodes de partitionnement est que les premières n'ont pas besoin de connaître le nombre de classes que l'on recherche pour les établir. En effet, une méthode hiérarchique, comme son nom l'indique, fourni comme résultat une hiérarchie de partitions emboîtées. Cela se représente graphiquement par un **arbre de classification**, dit **dendrogramme** (cf. Fig. 1.1).

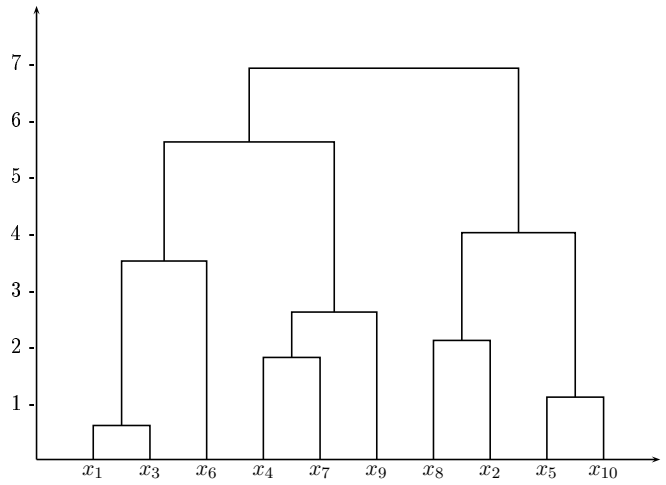


FIG. 1.1 – Exemple de dendrogramme avec 10 objets.

Au pied de l'arbre se trouve la partition discrète (où chaque observation est une classe), et au fur et à mesure que l'on monte en ordonnée (où est placée la valeur du critère $W(P,k)$), on regroupe les deux classes les plus proches, jusqu'à obtenir une seule classe contenant toutes les observations. On peut ensuite "couper" le dendrogramme à une certaine hauteur pour obtenir la partition en k classes. Comme on peut le voir sur la figure 1.2, pour obtenir 2 classes, on coupe le dendrogramme autour de l'ordonnée 6.3 ; ou de 4.5 pour en obtenir 3.

Il existe deux types de méthodes hiérarchiques, selon que l'on construit le dendrogramme en commençant par le bas en regroupant des classes, ou par le haut, en divisant les classes. Ces deux types d'approche seront détaillés dans la section suivante.

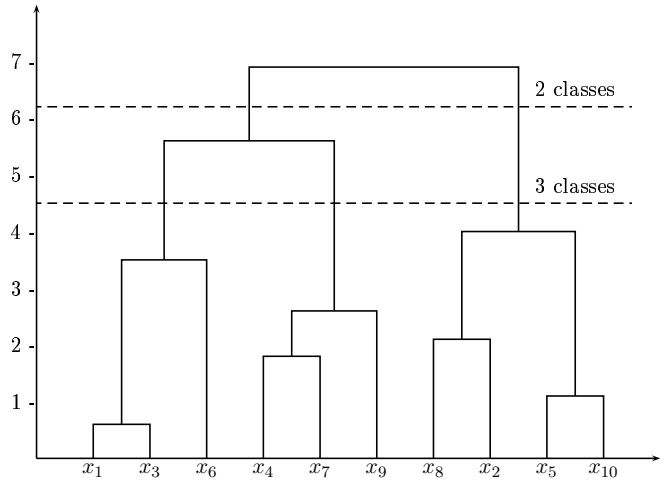


FIG. 1.2 – Exemple de l'extraction de classes d'un dendrogramme.

Les méthodes de partitionnement fonctionnent, elles, selon un autre principe. Elles cherchent parmi toutes les partitions en k classes (où k est fixé a priori) celle qui optimise le critère $W(P,k)$ prédéfini. Dans l'absolu, comme $\#(E)$ est fini, $\#\big(\mathcal{P}_k(E)\big)$ l'est aussi. On pourrait donc, en évaluant le critère pour chaque partition en k classes, trouver assurément la meilleure. Cependant, de par la nature combinatoire du problème, cette approche est totalement irréalisable. En pratique, on va générer une première partition en k classes puis, par un processus de réallocation des points, on va chercher à améliorer cette partition initiale.

1.3 Méthodes agglomératives ou divisives ?

Comme on l'a vu, les méthodes hiérarchiques se partagent en deux catégories. Elles peuvent, d'une part, être **agglomératives** (**ascendantes**), ou d'autre part **divisives** (**descendantes**). Dans le premier cas, elles commencent avec une partition où chaque observation est une classe, et elles regroupent à chaque étape les deux classes les plus "proches", jusqu'à ce qu'il n'y en ait plus qu'une seule. De la manière dont on définit que des classes sont "proches" découlent différentes méthodes de classification.

Voici quelques mesures de distances inter-classes, ainsi que la méthode de classification hiérarchique ascendante qui en découle :

- $d_{ls}(C_1, C_2) = \min_{(x_i \in C_1 \wedge x_j \in C_2)} d(x_i, x_j)$.
  $\longrightarrow$ méthode du **lien simple**, ou **méthode du plus proche voisin**.

- $d_{lc}(C_1, C_2) = \max_{(x_i \in C_1 \wedge x_j \in C_2)} d(x_i, x_j)$.
  $\longrightarrow$ méthode du **lien complet**, ou **du voisin le plus éloigné**

- $d_{lm}(C_1, C_2) = \frac{1}{n_1 \, n_2} \sum_{x_i \in C_1} \sum_{x_j \in C_2} d(x_i, x_j)$, avec $n_k = \#(C_k) \, \forall k$
  $\longrightarrow$ méthode du **lien moyen**.

Il existe encore bien d'autres méthodes ascendantes. En effet, celles-ci ont un grand avantage sur les méthodes descendantes : la complexité. Si on considère la complexité algorithmique des premières étapes pour les deux types de méthodes, on a :

- Méthodes ascendantes : il faut considérer tous les regroupements possibles de 2 individus parmi n. Ce qui représente $C_n^2 = \frac{n\,(n-1)}{2}$ possibilités. C'est donc une complexité polynomiale.

- Méthodes descendantes : il faut ici considérer toutes les divisions possibles de n observations en 2 groupes non vides. On peut donc avoir un point d'un côté et $n-1$ de l'autre, ou 2 points et $n-2$, etc. Ce qui nous donne $\frac{C_n^1 + C_n^2 + \ldots + C_n^n}{2} = 2^{n-1} - 1$. [2] On a donc affaire ici à une complexité exponentielle.

On comprend mieux que les méthodes ascendantes, plus rapides, soient plus communes. Cependant, les méthodes descendantes doivent effectuer moins d'itérations que les ascendantes pour obtenir un faible nombre de classes (ce qui est généralement recherché), vu qu'elles augmentent progressivement le nombre de classes en partant de 1, au lieu de le diminuer en partant de n.

[2] par le binôme de Newton : $(a+b)^n = \sum_{i=0}^{n} C_n^i \, a^i \, b^{n-i}$

Il existe cependant des méthodes descendantes qui parviennent à contourner cette difficulté, en optimisant un critère **local**, ou en travaillant monothétiquement (c-à-d en travaillant avec une seule variable à la fois, nous reviendrons plus en détail sur cette notion dans la section suivante). Citons pour l'exemple la Méthode de [Kaufman and Rousseeuw, 1990]. L'algorithme extrait de la plus "grande" classe le point dont la distance moyenne aux autres est la plus grande, et ensuite rattache à la nouvelle classe ainsi formée tous les points dont la distance moyenne aux points de leur classe est plus grande que la distance moyenne aux points de la nouvelle classe. Et une fois qu'il ne peut plus rajouter de point à la nouvelle classe ainsi formée, il recommence le processus avec la plus "grande" classe de la partition. On obtient ainsi une hiérarchie de partitions, créée de manière descendante. Cette méthode porte également le nom de **DIANA**.

1.4 Méthodes monothétiques ou polythétiques ?

Comme on l'a vu dans la section précédente, un moyen de réduire la complexité algorithmique des méthodes hiérarchiques descendantes est de travailler variable par variable. Cette approche porte le nom de **monothétique**. A contrario, si l'on travaille avec toutes les variables en même temps, on utilisera l'approche **polythétique**. La plupart des méthodes ascendantes, comme celles du lien simple, du lien complet ou du lien moyen, sont polythétiques. En effet, comme elles se basent sur une distance euclidienne, elles prennent en compte toutes les variables à la fois.

Plus formellement, on parle de classes monothétiques quand une conjonction de questions binaires permet de décider de l'appartenance d'un point à une classe. Une question binaire étant du type "$Y_j \in V$ ou $Y_j \notin V$?". En ordonnant les observations selon la variable Y_j, on peut réécrire cette question binaire sous la forme : "$Y_j \leq c$ ou $Y_j > c$?", où c est une valeur permettant de décrire l'appartenance au groupe V.

On découvre alors un autre avantage des méthodes monothétiques : la facilité d'interprétation. La classification ne se résume pas à établir des classes : il faut ensuite interpréter les résultats obtenus, expliquer en quoi les classes sont différentes les unes des autres. En quelque sorte leur "coller une étiquette". Et si on a travaillé variable par variable, on aura pour définir nos classes des expressions simples du type : " $Y_1 \leq 85$ et $Y_2 < 50$", comme on peut le voir avec l'arbre de classification de la figure 1.3.

Les méthodes monothétiques ont cependant des inconvénients. Si l'on se représente le nuage de point des données, les méthodes monothétiques effectuent uniquement des coupures parallèles aux axes, ce qui ne fonctionne pas (ne retourne pas la partition naturelle des données) avec tous les jeux de données (cf. fig. 1.4). De plus, on peut également avoir des observations proches, mais dont la variable de coupure diffère suffisamment que pour les mettre dans des classes différentes.

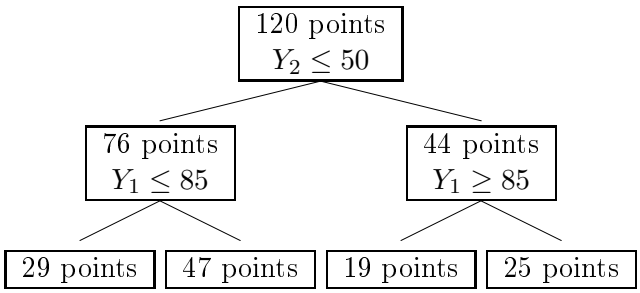


FIG. 1.3 – Exemple de résultats fournis par une méthode monothétique.

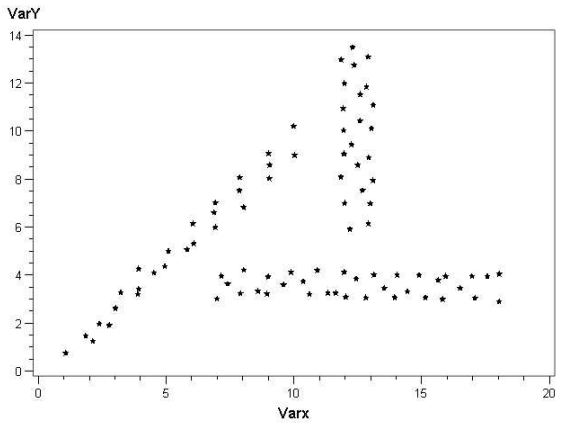


FIG. 1.4 – Jeu de données *"TRIANGLE"*, pour lequel une méthode monothétique aura des problèmes.

Chapitre 2

Modèles statistiques en classification automatique

Nous allons commencer par introduire le processus de Poisson. Nous verrons ensuite la méthode de classification des Hypervolumes, une méthode de partitionnement basée sur le processus de Poisson, et nous terminerons ce chapitre en présentant le Gap Test, un test d'hypothèse permettant de valider des classes.

2.1 Le processus de Poisson

Le processus de Poisson est un **processus ponctuel**. Un processus N est un processus ponctuel sur un domaine $D \subset \mathbb{R}^p$ si et seulement si N est une distribution aléatoire de points dans D telle que tout ensemble $A \subset D$ borné ne contienne qu'un nombre fini de points. On notera $N(A)$ la variable aléatoire qui compte le nombre de points dans A.

Le processus de Poisson peut se définir comme suit ([Cox and Isham, 1980]) : N est un **processus de Poisson de taux** λ sur un domaine $D \subset \mathbb{R}^p$ si, pour $A_1, A_2, ..., A_k \subset D$ des ensembles compacts, bornés et disjoints, les variables aléatoires $N(A_1), N(A_2), ..., N(A_k)$ sont indépendantes et ont respectivement des distributions de Poisson de paramètres $\lambda\, m(A_1)$, ..., $\lambda\, m(A_k)$, où $m(.)$ est la mesure de Lebesgue.

Plus précisément, N est un processus de Poisson si :

$$\begin{array}{c} \forall A_1, A_2, ..., A_k \subset D, \\ N(A_i) \perp\!\!\!\perp N(A_j), \ \textbf{et} \quad A_i \cap A_j = \phi \quad \forall i \neq j \in \{1, 2, ..., k\} \\ \\ \textbf{et} \\ \\ \forall A \subset D, \ \forall k > 0, \quad P\big(\, N(A) = k \,\big) \ = \ \frac{\big(\, \lambda\, m(A) \,\big)^k}{k!} \ e^{-\,\lambda\, m(A)} \end{array}$$

Comme on peut le voir dans [Rasson, 1976], le taux λ détermine complètement le processus. Il peut soit être constant, dans le cas du **processus de Poisson homogène**, ou **stationnaire**, soit dépendant des points de l'ensemble D, donnant alors le **processus de Poisson non homogène**, ou **non stationnaire**.

Citons une autre propriété du processus de Poisson : dans un ensemble de points issus d'un processus de Poisson, la probabilité que deux points coïncident est nulle. Remarquons aussi que les répartitions de points les plus aléatoires peuvent être modélisées par des processus de Poisson.

2.1.1 Le processus de Poisson homogène

Le processus de Poisson homogène jouit de la **propriété d'uniformité conditionnelle** : si $N(A) = n$ est fini, alors les n points sont indépendants et sont distribués aléatoirement et uniformément dans A.

Cela implique que si $x = \{x_1, ..., x_n\}$ est la réalisation d'un processus de Poisson homogène sur le domaine $D \subset \mathbb{R}^p$, cet ensemble de points a pour densité

$$f_X(x) \ = \ \frac{1}{m(D)} \ \ I_D(x)\,^1.$$

On peut donc obtenir ainsi la fonction de vraisemblance $\mathcal{L}$:

$$\mathcal{L}(D; x) \ = \ \prod_{i=1}^{n} f_X(x_i) \ = \ \frac{1}{\big(m(D)\big)^n} \ \prod_{i=1}^{n} I_D(x_i)$$

[1] $I_D(x) = \begin{cases} 1 & \text{si } x \in D \\ 0 & \text{sinon.} \end{cases}$

2.1.2 Le processus de Poisson non homogène

Comme on l'a vu, dans ce cas-ci, le taux λ n'est plus constant, mais dépend des points x de D. Il sera donc noté $\lambda(x)$.

Le **processus de Poisson non homogène** N d'intensité $\lambda(x)$ sur le domaine $D \subset \mathbb{R}^p$ $(0 < m(D) < \infty)$ est caractérisé par les deux propriétés suivantes :

- $\forall A \subset D$, $N(A)$ suit une distribution de Poisson de paramètre $\int_A \lambda(x)\, m(dx)$
- **Propriété conditionnelle de N** :
  Soit $N(A) = n$, alors les n points sont distribués aléatoirement et indépendamment dans A, avec une fonction de densité proportionnelle à $\lambda(x)$.

De la même manière qu'avec le cas homogène, nous obtenons la fonction de densité pour $x = \{x_1, ..., x_n\}$:

$$f_X(x) \;=\; \frac{\lambda(x)\, I_D(x)}{\int_D \lambda(t)\, m(dt)} \;=\; \frac{\lambda(x)\, I_D(x)}{\rho(D)};$$

avec $\rho(D) = \int_D \lambda(t)\, m(dt)$, l'**intensité intégrée** du processus sur D.

Et donc, la fonction de vraisemblance vaut :

$$\mathcal{L}(D; x) \;=\; \prod_{i=1}^{n} f_X(x_i) \;=\; \frac{1}{\big(\rho(D)\big)^n} \;\prod_{i=1}^{n} I_D(x_i)\, \lambda(x_i).$$

Estimation de l'intensité du processus

Il existe différentes méthodes destinées à estimer l'intensité d'un processus de Poisson non homogène. Nous allons brièvement en voir deux : **les histogrammes** et **les noyaux**.

Commençons par les **histogrammes** ([Delecroix, 1983]).
Un histogramme est un graphique constitué d'une suite de rectangles juxtaposés. Leur base représente la largeur des classes (en abscisse), et leur surface leur fréquence (et donc leur effectif). On a des **histogrammes réguliers** ou

à amplitudes égales dans le cas où toutes les classes ont la même largeur. La hauteur des rectangles est alors également proportionnelle à leurs effectifs. Si les classes sont de largeurs variables, on parle alors d'**histogrammes à fréquences égales**.

Cette représentation implique qu'on représente une fréquence f_k, observée sur un intervalle $]e_{k-1}; e_k]$ de longueur a_k d'une variable continue, par une aire. On doit donc effectuer un regroupement des données, qui nous fait perdre de l'information sur la distribution de la variable à l'intérieur d'une classe. Il faut alors effectuer une hypothèse de distribution interne ; la plus simple étant la distribution uniforme à l'intérieur de chaque classe.

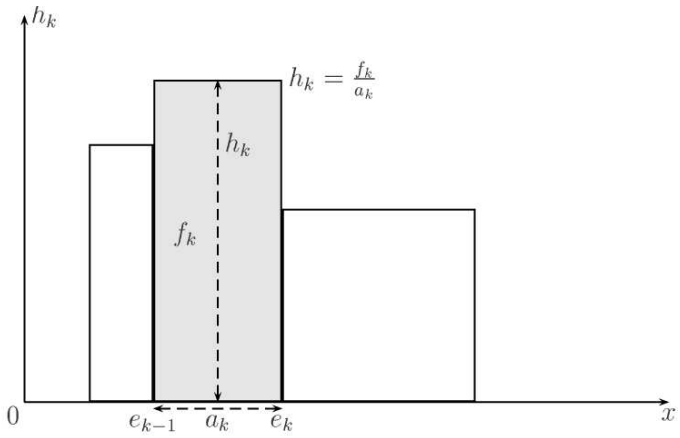


FIG. 2.1 – Représentation d'un histogramme ([Bouget and Viénot, 1983]).

Pour construire l'histogramme, il faut commencer par choisir le nombre de classes. Dans sa thèse, [Pirçon, 2004] a opté pour un nombre de classes théorique n_{th} tel que $2^{n_{th}-1} < n < 2^{n_{th}}$, avec n le nombre total de points. Ensuite, il faut délimiter les classes. Dans le cas des histogrammes réguliers, il suffit de partager l'intervalle de valeurs de la variables en n_{th} sous-intervalles de même longueur. Remarquons que cette méthode est assez simple, mais il est possible d'avoir des classes qui ne contiennent pas de points. Pour construire un histogramme à fréquences égales, il faut calculer un nombre théorique $n_{pt_{th}}$ de points dans chaque classe ($n_{pt_{th}} = \frac{n}{n_{th}}$, ajusté pour que $n_{pt_{th}}$ soit entier).

Voici une autre méthode : celle des **noyaux** ([Silverman, 1986]). Nous ne travaillerons ici qu'en dimension 1, puisque les méthodes dont nous parlerons par la suite sont monothétiques. Considérons tout d'abord un **estimateur naïf**. Il estime la densité f de la variable X par

$$f(x) = \lim_{h \to 0} \frac{1}{2h} P\big(x - h < X < x + h\big).$$

On définit une fonction K de poids par

$$K(x) = \begin{cases} \frac{1}{2} & si\ |x| < 1 \\ 0 & sinon \end{cases},$$

et on peut écrire l'estimateur naïf comme étant

$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h} K\left(\frac{x - x_i}{h}\right).$$

Comme mentionné dans [Pirçon, 2004, page 51], "*l'estimateur est construit en plaçant une "fenêtre" de largeur* $2h$ *et de hauteur* $\frac{1}{2nh}$ *sur chaque observation et en sommant toutes les observations. Chaque point de l'intervalle* $]x-h, x+h[$ *contribue donc pour une quantité* $\frac{1}{2nh}$ *à l'estimation en* x *de la densité. Par contre, pour tout point extérieur à cet intervalle, la contribution est nulle.*" La figure 2.2 illustre ce principe.

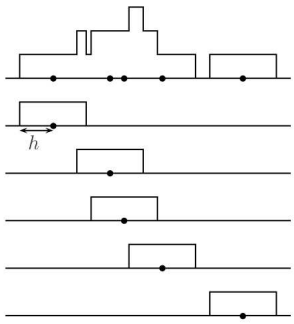


FIG. 2.2 – Fonctionnement de l'estimateur naïf ([Daudin et al., 1988]).

Un inconvénient de cet estimateur est qu'il n'est pas continu. On voit bien, en effet, que la fonction possède des sauts. On va donc prendre une autre fonction de poids K telle que

$$\int_{-\infty}^{+\infty} K(x)\,dx = 1, \quad \text{avec K symétrique et continu.}$$

On aura alors ce qu'on appelle la **méthode des noyaux**. Un choix possible pour la fonction de poids est de prendre la loi normale :

$$K(x) \;=\; \frac{1}{\sqrt{2\pi}}\, e^{-\frac{y^2}{2}}.$$

L'estimateur est alors défini par

$$\hat{f}(x) \;=\; \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h}\, K\left(\frac{x - X_i}{h}\right).$$

Comme on peut le voir sur la figure 2.3, la méthode des noyaux fonctionne en superposant des bosses là où l'estimateur naïf utilisait des boîtes. L'estimateur des noyaux est donc continu.

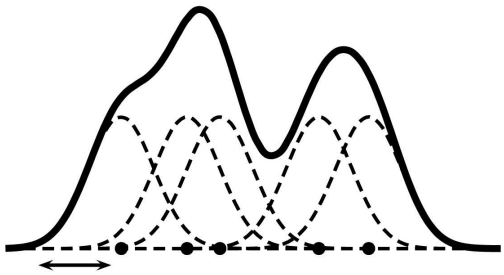


FIG. 2.3 – Estimation par le noyau normal ([Pirçon, 2004]).

Remarquons encore l'importance du paramètre h, appelé **paramètre de lissage** ([Akaike, 1954], [Rosenblatt, 1956], [Parzen, 1962]), qui intervient aussi bien dans le cas de l'estimateur naïf que dans celui des noyaux. Il détermine la largeur des bosses (ou des boîtes de l'estimateur naïf). S'il est trop petit, on obtiendra n pics au niveau des observations, et s'il est trop grand, on n'aura alors plus qu'une seule bosse. Il faut donc le choisir judicieusement ! *Silverman* a montré que "*le nombre de modes est une fonction décroissante de h*" dans le cas du noyau normal ([Silverman, 1986]). Il conseille de prendre pour h la plus grande valeur possible telle que $\hat{f}$ reste multimodale ([Silverman, 1981]).

Passage du cas non homogène au cas homogène

Remarquons qu'il est possible de passer du processus de Poisson non homogène au processus de Poisson homogène par un simple changement de variables ([Cox and Isham, 1980, page 48]).

Soit $\{x_1, ..., x_n\}$ une réalisation d'un processus de Poisson non homogène d'intensité $\lambda(x)$ sur $\mathbb{R}^p$. Appliquons-lui le changement de variables suivant :

$$\tau(x) = \int_0^x \lambda(t)\, m(dt).$$

On obtient alors $\{\tau_1, ..., \tau_n\}$[2], qui suit un processus de Poisson homogène.

2.2 Modèles basés sur le processus de Poisson homogène

Nous allons aborder ici la méthode de classification des Hypervolumes ([Hardy and Rasson, 1982]) dans le cas du processus de Poisson homogène. Pour ce faire, nous commencerons par nous pencher sur le problème de l'estimation d'un ensemble convexe. La section suivante se contentera de généraliser au cas non homogène les résultats présentés dans cette section.

[2] avec $\tau_i = \tau(x_i) \quad \forall i = 1, 2, ..., n$

2.2.1 Estimation d'un ensemble convexe

Le problème de base que l'on va rencontrer est l'estimation d'un ensemble convexe :
"Etant donné la réalisation d'un processus de Poisson homogène dans un domaine convexe compact D, estimer D en utilisant des méthodes d'inférence statistique" [Ripley and Rasson, 1977]

La solution a été apportée par [Ripley and Rasson, 1977] et [Rasson, 1978]. Notons C l'ensemble des points de D, et $H(C)$ l'enveloppe convexe de C. $H(C)$ est l'estimateur du *maximum de vraisemblance* de D. C'est également une *statistique exhaustive* pour D, mais qui est *biaisée* : $H(C)$ sera toujours plus petit que D.
Pour obtenir un estimateur non biaisé, il faut effectuer une dilatation de l'enveloppe convexe, à partir du centre de gravité des points sur l'enveloppe convexe, selon un coefficient de dilatation approximé par

$$c = \sqrt{\frac{n}{n - V_n}} \ ,$$

où V_n représente le nombre de points sur l'enveloppe convexe ([Moore, 1984]).

2.2.2 La méthode de classification des Hypervolumes

La méthode de classification des Hypervolumes est une méthode de k-partitionnement (avec k fixé à l'avance). Elle doit donc retrouver la meilleure partition en k classes P^*, au sens d'un certain critère $W(P, k)$.

On part de l'hypothèse selon laquelle les n points que l'on veut classer sont issus d'un processus de Poisson homogène dans $D \subset \mathbb{R}^p$, D étant l'union de $D_1, D_2, ..., D_k$, k domaines **convexes disjoints inconnus**, et on veut estimer ces domaines D_i.

On recherche pour cela la fonction de vraisemblance $\mathcal{L}$:

$$\begin{array}{c} \mathcal{L}(D; x_1, x_2, ..., x_n) \ = \ f(x_1, x_2, ..., x_n; D) \ = \ \prod_{i=1}^{n} f_{X_i}(x_i; D) \\ = \prod_{i=1}^{n} \frac{1}{m(D)} \ I_D(x_i) \ = \ \frac{1}{\big(m(D)\big)^n} \ I_D\big(H(x_1, x_2, ..., x_n)\big), \end{array}$$

avec $H(x_1, x_2, ..., x_n)$ l'enveloppe convexe des n points de l'échantillon.

Comme son nom l'indique, le principe du maximum de vraisemblance impose de rechercher le maximum de la fonction de vraisemblance, en fonction des domaines D_i :

$$\max_{D_1,...,D_k} \mathcal{L}(D;x) \Leftrightarrow \min_{P\in\mathcal{P}_k} \sum_{i=1}^{k} m\big(H(C_i)\big) \Leftrightarrow \min_{P\in\mathcal{P}_k} W_k.$$

Les estimateurs du maximum de vraisemblance des domaines inconnus D_i sont les enveloppes convexes $H(C_i)$ telles que $\sum_{i=1}^{k} m\big(H(C_i)\big)$ soit minimale. Le critère des Hypervolumes est donc défini par :

$$W_k : \mathcal{P}_k \longrightarrow \mathbb{R}^+ : P \mapsto W(P,k) = \sum_{i=1}^{k} m\big(H(C_i)\big) = \sum_{i=1}^{k} \int_{H(C_i)} m(dx).$$

2.3 Modèles basés sur le processus de Poisson non homogène

Cette généralisation de la méthode des Hypervolumes a été faite par [Rasson and Granville, 1996]. Les points sont maintenant générés par un processus de Poisson non homogène N d'intensité $\lambda(.)$, dans un domaine $D \subset \mathbb{R}^p$, qui est toujours l'union de k domaines convexes disjoints D_i inconnus, qu'il faut estimer. Si $\lambda(.)$ est lui aussi inconnu, il faut alors commencer par l'estimer.

Comme dans le cas du processus de Poisson homogène, on va rechercher la fonction de vraisemblance $\mathcal{L}$:

$$\mathcal{L}(D;x_1,...x_n) = \prod_{i=1}^{n} f_{X_i}(x_i) = \frac{1}{\big(\rho(D)\big)^n} \prod_{i=1}^{n} I_D(x_i)\,\lambda(x_i).$$

Et comme :

$$\max_{D_1,...D_k} \mathcal{L}(D;x) \Leftrightarrow \min_{P\in\mathcal{P}_k} \sum_{i=1}^{k} \rho\big(H(C_i)\big) \Leftrightarrow \min_{P\in\mathcal{P}_k} W_k$$

On obtient alors le critère généralisé des Hypervolumes :

$$\tilde{W}_k = \sum_{i=1}^{k} \rho\big(H(C_i)\big) = \sum_{i=1}^{k} \int_{H(C_i)} \lambda(x)\; m(dx).$$

2.4 Le Gap Test

Le **Gap Test** ([Kubushishi, 1996]) est un test d'hypothèse qui permet de valider des classes. Il ne s'applique qu'à des données issues d'un processus de Poisson homogène, mais avec le changement de variables vu précédemment(cf. 2.1.2), on peut l'utiliser aussi dans le cas non homogène.

Voyons le principe de ce test. On a deux domaines D_1 et D_2, contenant respectivement n_1 et n_2 points issus d'un processus de Poisson homogène. On veut savoir si ces $n = n_1 + n_2$ points appartiennent bien aux domaines D_1 et D_2, ou s'ils appartiennent à un même domaine D. Les domaines D, D_1 et D_2 sont inconnus, et on note C, C_1 et C_2 l'ensemble des points appartenant respectivement à D, D_1 et D_2.

On teste alors l'hypothèse H_0 contre l'hypothèse H_1 définies par :
- H_0 : les $n = n_1 + n_2$ points sont issus d'un même processus de Poisson homogène dans le domaine D ;
- H_1 : les n_1 points sont issus d'un processus de Poisson homogène dans le domaine D_1, et les n_2 points dans D_2, avec $D_1 \cap D_2 = \phi$.

Recherchons la région critique du test. Par la méthode du quotient de vraisemblance, on obtient ([Kubushishi, 1996, page 65]) :

$$Q(x) \;=\; \left(1 - \frac{m(\Delta)}{m(D)}\right)^n$$

avec $\Delta = H(C) \;\setminus\; \big(H(C_1) \cup H(C_2)\big)$, le "trou" entre les deux classes, $m(.)$ étant toujours la mesure de Lebesgue.

On a donc une région critique du type : $\left(1 - \frac{m(\Delta)}{m(D)}\right) \leq c$ où c est une constante. [Kubushishi, 1996] a montré qu'on obtenait la règle de décision suivante (dans le cas de distributions asymptotiques) :

on rejette H_0 au niveau α si

$\frac{n\,m(\Delta)}{m\big(H(C)\big)} \;-\; log(n) \;\geq\; -log\big(-log(1-\alpha)\big)$, en dimension 1 ;

$\frac{n\,m(\Delta)}{m\big(H(C)\big)} \;-\; log(n) \;-\; (p-1)\,log\big(log(n)\big) \;\geq\; -log\big(-log(1-\alpha)\big)$, en dimension p.

Chapitre 3

Cinq méthodes de classification monothétiques basées sur les processus de Poisson

Ce chapitre va détailler le fonctionnement des cinq nouvelles méthodes de classification de [Pirçon, 2004], à savoir *HOPP, SONHOPPHI, SONHOPPKI, UNHOPPHI* et *UNHOPPKI*.

Ces méthodes sont constituées de 3 étapes, la première créant un arbre de classification, la seconde élaguant cet arbre, et la dernière essayant de recoller les feuilles qui peuvent l'être. Ces méthodes sont, dans un premier temps, hiérarchiques divisives monothétiques. Elles effectuent des divisions de l'ensemble des observations, selon une variable à la fois, de manière à obtenir une hiérarchie de partitions sous forme d'arbre (binaire). Mais avec l'étape de recollement, toute la hiérarchie de l'arbre peut être bouleversée. Au final, nous n'avons donc plus des méthodes hiérarchiques, mais des méthodes de partitionnement !

Comme nous l'avons vu, l'intérêt des méthodes divisives monothétiques est la facilité d'interprétation des résultats, puisque chaque classe peut s'expliquer par la conjonction des critères de séparation des noeuds. Et les classes obtenues après recollement conservent cette qualité, ce qui est un atout non négligeable !

Ces cinq méthodes se basent sur le processus de Poisson. La première, *HOPP*, sur le processus de Poisson homogène, les autres sur le processus de Poisson non homogène. Il faut donc, pour ces quatre autres méthodes, estimer l'intensité du processus, ce qui se fait par les histogrammes, ou par les noyaux. Cette estimation peut se faire avant de commencer à couper, on suppose donc que les points sont issus d'un seul processus de Poisson, ou bien à chaque coupure, supposant alors qu'il y a superposition de processus.

Définissons ces 5 méthodes :

- *HOPP* : les points sont issus d'un processus de Poisson homogène ;
- *UNHOPPHI* : les points sont issus d'un processus de Poisson non homogène, avec une estimation de la densité par la méthode des histogrammes ;
- *UNHOPPKI* : les points sont issus d'un processus de Poisson non homogène, avec une estimation de la densité par la méthode des noyaux ;
- *SONHOPPHI* : les points sont issus d'une superposition de processus de Poisson non homogènes, avec une estimation de la densité par la méthode des histogrammes ;
- *SONHOPPKI* : les points sont issus d'une superposition de processus de Poisson non homogènes, avec une estimation de la densité par la méthode des noyaux.

Nous allons maintenant étudier les 3 étapes de ces méthodes.

3.1 Création de l'arbre

La première étape est la création de l'arbre. Cela se fait de façon hiérarchique descendante monothétique. Nous allons commencer par voir comment on peut couper un noeud, puis nous énoncerons quelques critères pour établir si un noeud est terminal (feuille) ou non. Nous verrons ensuite comment choisir le noeud à couper, et nous terminerons par la présentation des algorithmes régissant l'élaboration de l'arbre.

3.1.1 Coupure d'un noeud

Commençons par expliquer comment l'algorithme coupe un noeud. Il utilise le critère du *maximum de vraisemblance*. Comme on l'a vu précédemment, avec un ensemble $x = \{x_1, ..., x_n\}$ issu d'un processus de Poisson sur un domaine D, la fonction de vraisemblance $\mathcal{L}_D$ vaut :

$$\mathcal{L}(D;x) = \begin{cases} \dfrac{1}{\big(m(D)\big)^n} \displaystyle\prod_{i=1}^{n} I_D(x_i) & \text{dans le cas homogène ;} \\ \dfrac{1}{\big(\rho(D)\big)^n} \displaystyle\prod_{i=1}^{n} I_D(x_i)\,\lambda(x_i) & \text{dans le cas non homogène ;} \end{cases}$$

avec $\lambda(.)$ l'intensité du processus de Poisson non homogène, $\rho(.)$ l'intensité intégrée $\big(\rho(D) = \int_D \lambda(x)\,dx\big)$ et $m(.)$ la mesure de Lebesgue.

Comme le produit ne dépend pas de D, on a que

$$\max_D \mathcal{L}(D;x) \iff \begin{cases} \min\limits_{\{D \,|\, x_1,...,x_n \,\in D\}} m(D) & \text{dans le cas homogène ;} \\ \min\limits_{\{D \,|\, x_1,...,x_n \,\in D\}} \rho(D) & \text{dans le cas non homogène .} \end{cases}$$

Maximiser la fonction de vraisemblance revient donc à minimiser la "taille" $\big(m(.)$ ou $\rho(.)$ selon le cas, que nous allons noter $l(.)\big)$ de D. Comme on veut partitionner D en 2 parties convexes D_1 et D_2, on va remplacer la mesure de D par la somme des mesures de D_1 et D_2. L'intérêt est que $D_1 \cup D_2$ contient tous les points de D, mais comme on peut le voir sur la figure 3.1,

$$l(D) \;=\; l(D_1) + l(\Delta) + l(D_2) \;\geq\; l(D_1) + l(D_2).$$

On va donc rechercher à maximiser ce "trou" Δ !

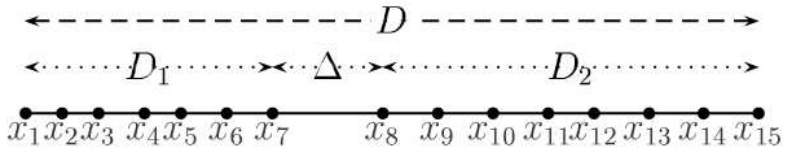


FIG. 3.1 – Coupure de D en D_1 et D_2 ([Pirçon, 2004]).

La règle de coupure de [Pirçon, 2004] est donc :

> *"trouver la variable pour laquelle le trou engendré dans les données selon cette variable est le plus grand selon la mesure utilisée ($m(.)$ si le processus de Poisson est homogène ou $\rho(.)$ si le processus de Poisson est non homogène)."*

Plus simplement, il suffit, pour chaque variable, de trier les observations par ordre croissant et rechercher le plus grand écart entre deux points consécutifs, puis de retourner le maximum ainsi que la variable qui a permis de l'atteindre.

Remarquons que, pour être moins sensible aux outliers, il est possible de travailler à l'intérieur de la **barrière de Tuckey**, [Jobson, 1991, pages 56-59] :

$$[Q_1 - 1.5 * Q\,,\, Q_3 + 1.5 * Q],$$

avec Q_1 le premier quartile, Q l'écart inter-quartile, et Q_3 le troisième quartile.

3.1.2 Arrêt des coupures

Il y a tout d'abord quelques règles logiques qui déterminent si un noeud est une feuille ou non :

- 1. les points du noeud ont tous les mêmes coordonnées ;
- 2. le noeud contient très peu de points (moins de 10% de l'effectif total et/ou moins de 10) ;
- 3. le trou Δ enlevé est insignifiant (de mesure inférieure au seuil 10^{-5} fixé).

On peut également envisager d'effectuer un **test de Fisher** pour voir si la coupure de la classe C contenant n points en les classes C_1 et C_2 est significative ou pas ([Pirçon and Rasson, 2002]). Celui-ci compare les variances des groupes en testant l'hypothèse H_0 contre l'hypothèse H_1, avec

$$H_0 :\ \sigma_C^2 = \sigma_{C_1+C_2}^2 \qquad \text{et} \qquad H_1 :\ \sigma_C^2 \neq \sigma_{C_1+C_2}^2.$$

La statistique de ce test est

$$F = \frac{SS_C \ /\ (n-1)}{\left(SS_{C_1} + SS_{C_2}\right) \ /\ (n-2)},$$

où SS_C est la *sum of squared differences* de la classe C.

Et la région critique (RC), telle qu'un noeud est final si $x \notin$ RC :

$$\left\{ x \,|\, F \geq F_{n-1\,,\,n-2\;;\;0{,}025} \right\}$$

où $F_{n-1\,,\,n-2\;;\;0{,}025}$ est le quantile d'ordre 0.025 d'une loi de Fisher-Snedecor à $n-1$ degrés de liberté au numérateur, et $n-2$ au dénominateur.

3.1.3 Choix du noeud à couper

Maintenant que nous savons comment couper un noeud, et jusqu'à quand le couper, étudions le choix du noeud qu'il nous faut couper. Comme à chaque coupure, on calcule la mesure du trou enlevé, un critère simple, et qui est en accord avec le principe du maximum de vraisemblance, est de prendre le noeud qui permet d'ôter le plus grand trou.

3.1.4 Algorithmes

Voici l'algorithme de choix du noeud à couper, ainsi que celui de la coupure d'un noeud.

- **Pour chaque noeud terminal ($i = 1, \ldots, k$) de l'arbre, faire :**
  - Calculer par l'algorithme 3.2 la mesure du trou maximum Δ obtenu si c'est le noeud i que l'on coupe :

    $$\Delta_{(i)} = \Delta \text{ de l}'\text{algorithme 3.2 de coupure appliqué au noeud } i\ .$$

- **Fin de la boucle.**
- Choix du meilleur noeud à couper :

  $$\text{noeud}_{max} = \arg \max_{i=1,\ldots,k} \Delta_{(i)}\ .$$

- Coupure de noeud$_{max}$ par l'algorithme 3.2.
- Mise à jour de l'arbre (noeud$_{max}$ partitionné).
  L'arbre contient alors $k+1$ feuilles.

ALGORITHME 3.1 : Algorithme du choix du groupe à couper.

Soit C un noeud de l'arbre.
Notons x les n points contenus dans C.

- Si n est inférieur à 10% de l'effectif initial et/ou inférieur à 10, alors C est une feuille ;
- Si tous les points x_i sont égaux, alors C est une feuille ;
- Sinon :
  - **Pour chaque variable ($j = 1, \ldots, p$), faire :**
    - Si le processus de Poisson est non homogène, estimer la densité selon la variable j ;
    - Calculer les statistiques d'ordre des points x_i selon la variable j :

$$(x_{(1),j}\ ,\ \ldots\ ,\ x_{(n),j})\ ;$$

    - Rechercher le plus grand trou $\delta_{max}(j)$:

$$\delta_{max}(j) = \max_{i=b_{inf},\ldots,b_{sup}-1} l(x_{(i),j}\ ,\ x_{(i+1),j})$$

$$i_{max}(j) = \arg \max_{i=b_{inf},\ldots,b_{sup}-1} l(x_{(i),j}\ ,\ x_{(i+1),j})$$

    où b_{inf} et b_{sup} sont les bornes inférieure et supérieure de la barrière de Tukey ($1 \leq b_{inf} \leq b_{sup} \leq n$), et $l(.,.)$ est la distance euclidienne $d(.,.)$ dans le cas homogène, ou la distance "intensité intégrée" $\rho(.,.)$ dans le cas non homogène.
  - **Fin de boucle.**
  - Trouver la variable donnant le plus grand trou :

$$var_{max} = \arg \max_{j=1,\ldots,p} \delta_{max}(j)\ .$$

    Le trou maximum Δ vaut : $\delta_{max}(var_{max})$.
  - Mise à jour éventuelle :
    - Si Δ est inférieure à 10^{-5}, alors C est une feuille ;
    - Sinon couper le domaine D en D_1 et D_2 tels que

$$\{x_{(1),var_{max}}, \ldots, x_{(i_{max}(var_{max})),var_{max}}\} \in D_1\ ,$$
$$\{x_{(i_{max}(var_{max})+1),var_{max}}, \ldots, x_{(n),var_{max}}\} \in D_2\ .$$

      Les deux fils C^1 et C^2 de C sont alors respectivement constitués des points :

$$\overline{x}_1 = (x_{(1)}, \ldots, x_{(n_1)})\ ,$$
$$\overline{x}_2 = (x_{(n_1+1)}, \ldots, x_{(n)})$$

      où $(x_{(1)}, \ldots, x_{(n)})$ sont les statistiques d'ordre de x selon la variable var_{max}, et $n_1 = i_{max}(var_{max})$.

ALGORITHME 3.2 : Algorithme de coupure.

3.2 Elagage de l'arbre

Une fois l'arbre de classification obtenu, il nous faut l'élaguer. En effet, il se peut que l'on ait coupé un noeud qui ne devait pas l'être (mauvaise coupure). Il se peut aussi que des mauvaises coupures précèdent des coupures judicieuses (bonnes). C'est pourquoi on ne teste pas la coupure au moment de la faire, mais une fois que l'arbre est terminé. La figure 3.2 illustre bien qu'une mauvaise coupure peut être nécessaire pour en obtenir de bonnes.

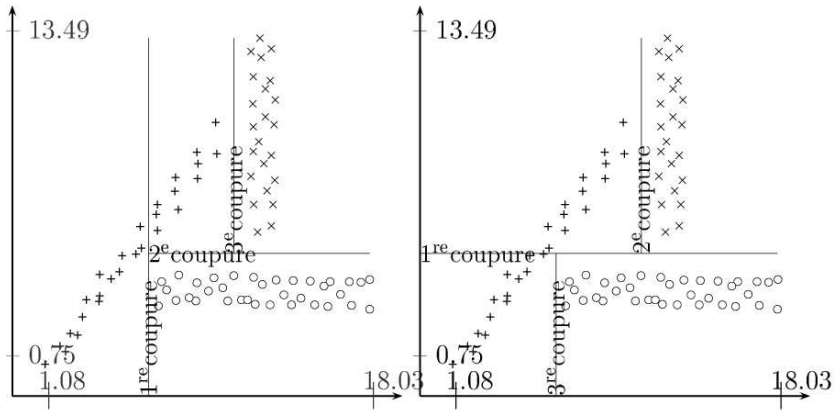


FIG. 3.2 – Exemples de mauvaises coupures permettant d'en obtenir des bonnes ([Pirçon, 2004]).

Il nous faut donc un critère permettant d'établir si une coupure est bonne ou non. Nous allons voir deux méthodes : le **Gap Test**, et l'**Inertie**.

3.2.1 Elagage par le Gap Test

Nous allons appliquer ici le test d'hypothèse du Gap Test (cf. 2.4, page 19). Comme nous l'avons vu précédemment, ce test ne s'applique qu'à des données issues d'un processus de Poisson homogène, et donc, dans le cas non homogène, il faudra effectuer un changement de variables (cf. 2.1.2, page 16) pour retomber dans le cas homogène.

On effectue donc le test de H_0 (un seul groupe) contre l'hypothèse alternative H_1 (deux groupes disjoints) sur chaque noeud de l'arbre. Chaque noeud pour lequel l'hypothèse H_0 n'est pas rejetée n'aurait donc pas dû être coupé, tandis que ceux qui rejettent H_0 contiennent bien deux groupes distincts. Cela nous permet donc de définir pour chaque coupure si elle est bonne ou mauvaise.

Comme une mauvaise coupure peut donner suite à de bonnes coupures, on aura pour stratégie d'élagage de couper tous les bouts de branches qui ne contiennent que des mauvaises coupure. La figure 3.3 illustre cette stratégie d'élagage sur un arbre dont chaque coupure est déjà caractérisée.

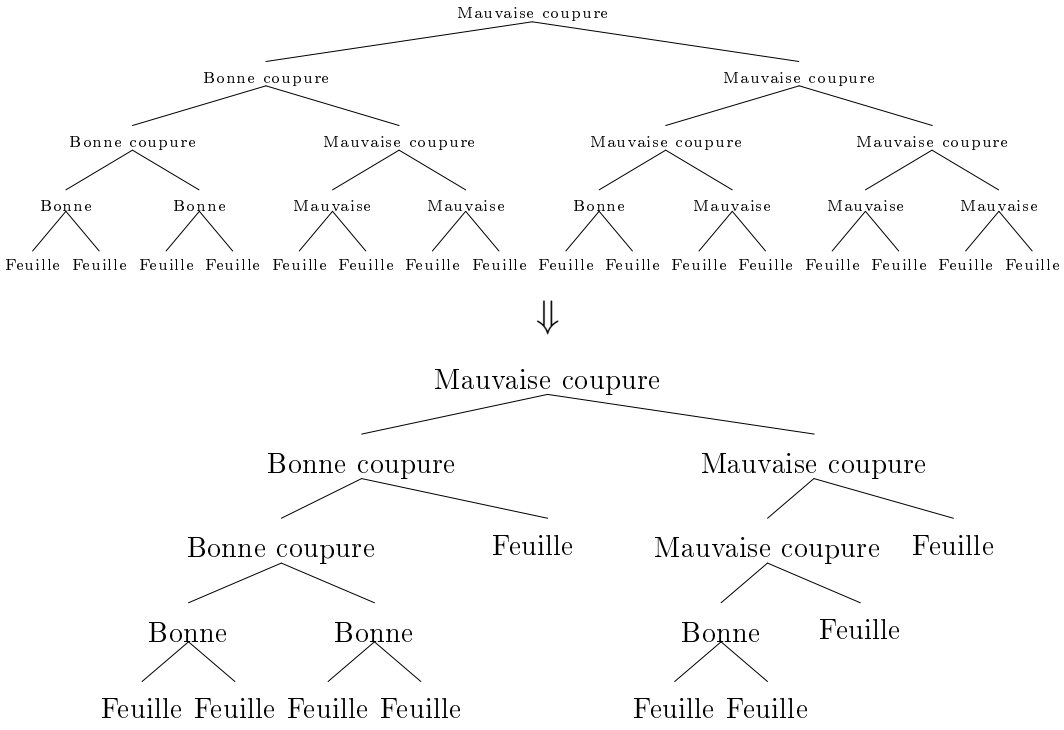


FIG. 3.3 – Illustration du principe d'élagage par le Gap Test.

Remarquons qu'il existe deux types de Gap Test. Nous pouvons utiliser les lois exactes, dont le coût algorithmique est assez élevé, ou les lois limites. En pratique, la loi exacte ne sera utilisée qu'avec moins de 100 observations.

Voici l'algorithme commun aux deux lois :

Soit $\alpha = 0.05$, le niveau du test.
On suppose l'arbre de classification construit.
- **Pour chaque coupure d'un noeud de l'arbre, faire :**
  - Soit la coupure du noeud C (n points) en ses fils C^1 (n_1 points) et C^2 (n_2 points), qui s'est faite selon la variable var_{max}.
  - Si le processus de Poisson est non homogène :
    - Estimer l'intensité $\lambda(t)$ du processus pour les n points selon la variable var_{max} ;
    - Transformer les données par le changement de variables vu en 2.1.2 (page 16) pour qu'elles soient des réalisations d'un processus de Poisson homogène :
$$\tau(x) = \int_0^x \lambda(t)\, dt\ .$$
  - Calculer la mesure du trou Δ et la mesure du domaine D des n points.
$$\rightarrow\ m(\Delta)\,,\ m(D)$$
  - Si on travaille avec les lois exactes et si $n \leq 100$:
    - Trouver c tel que $\displaystyle\sum_{k=0}^{n} (-1)^k \binom{n}{k} \Big(\max(1-kc, 0)\Big)^{n-1} = 1-\alpha$;
    - Si $\dfrac{m(\Delta)}{m(D)} \leq c$, alors la coupure est déclarée "mauvaise" ;
    - Sinon, la coupure est déclarée "bonne".
  - Sinon :
    - Si $\dfrac{nm(\Delta)}{m(D)} - \log n \leq -\log\big(-\log(1-\alpha)\big)$, alors la coupure est déclarée "mauvaise" ;
    - Sinon, la coupure est déclarée "bonne".
- **Fin de la boucle.**
- Elagage de tous les bouts de branches qui ne contiennent que des mauvaises coupures.

ALGORITHME 3.3 : Algorithme d'élagage par le Gap Test.

3.2.2 Elagage par l'inertie

Nous présentons ici un autre critère, basé sur l'**inertie**, différent de celui qui nous a permis d'établir l'arbre. Comme [Jobson, 1992] l'a expliqué, dans le cas de méthodes ascendantes, la sélection d'une étape de la hiérarchie doit se faire par un *second* critère d'optimalité. De même, comme l'élagage permet de fixer le nombre de classes, on peut se permettre d'utiliser un autre critère.

Ce critère consiste à maximiser $W(.)$ l'**inertie intra-classe**, qui est une mesure de la compacité des classes d'une partition.

Soit une partition $\mathcal{P}_k = \{C_1, C_2, ..., C_k\}$; l'inertie intra-classe est définie par :

$$W(\mathcal{P}_k) = \sum_{j=1}^{k} I(C_j),$$

où $I(C_j)$ est l'inertie de C_j, telle que

$$I(C) = \frac{1}{2\ \#(C)} \sum_{x_i \in C} \sum_{x_j \in C} d(x_i, x_j) = \sum_{x_i \in C} d(x_i, g),$$

où $g = \frac{1}{\#(C)} \sum_{x_i \in C} x_i$ représente le centre gravité de C, et $d(.,.)$ est la distance euclidienne.

Remarquons que ce critère a pour biais de privilégier des classes hypersphériques.

Nous utiliserons ce critère de la façon suivante : chaque coupure d'un noeud crée une partition en $k+1$ classes à partir de la partition en k classes obtenue précédemment. En suivant l'ordre de la hiérarchie, on obtient donc la meilleure partition en 1, 2, ..., k classes. On associe à chaque partition sa valeur d'inertie intra-classe, qu'on représente sous forme d'un graphe ayant en abscisse le nombre de classes de la partition et en ordonnée la valeur du critère. Le nombre de classes optimal sera donné par la présence d'un "coude" dans le graphique.

Cette méthode de détermination du nombre de classes porte le nom de **méthode du coude**.

Un exemple de la détermination du nombre de classes est donné par la figure 3.4, où l'on voit bien une cassure de la courbure du graphe au niveau de l'abscisse 4, ce qui indique que la meilleure partition est celle en 4 classes. Cette méthode trouve sa justification dans le fait que l'inertie diminue toujours avec l'augmentation du nombre de classe (elle est même nulle dans le cas où il n'y a qu'un point par classe). Le coude du graphique indique que l'augmentation du nombre de classe n'apporte plus de baisse significative de l'inertie, et qu'il n'est plus intéressant d'augmenter le nombre de classes.

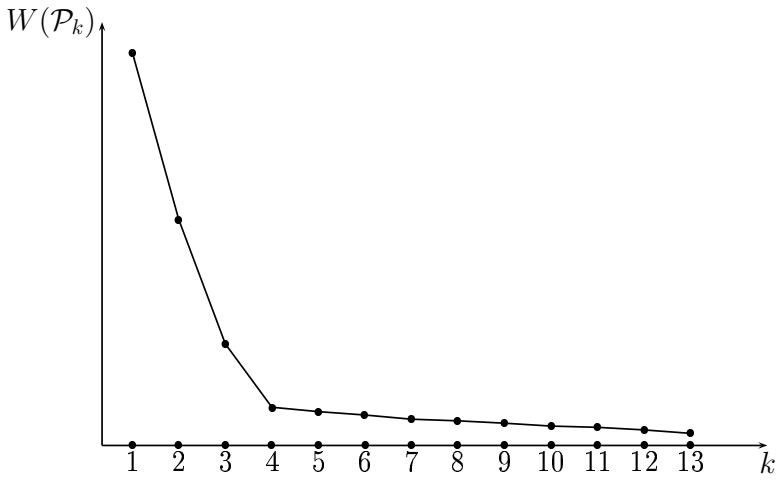


FIG. 3.4 – Illustration de la détermination du nombre de classes par la méthode du coude.

Pour élaguer l'arbre, une fois le nombre k de classes à retourner connu, il suffit de supprimer toutes les branches issues de coupures "dépassant" de la partition en k classes. D'un point de vue pratique, si on a numéroté les noeuds dans l'ordre de leur création, en commençant par 1 à la racine, il suffit de supprimer tous ceux dont le numéro est supérieur à $2k - 1$.

Remarquons que la recherche de coudes est tout à fait visuelle, et doit donc être faite par l'utilisateur. C'est pourquoi dans l'algorithme de la méthode (Algorithme 3.4), il faut tracer le graphe.

Voici l'algorithme d'élagage par la méthode d'inertie :

On suppose l'arbre de classification construit.
Soit $I(\cdot)$ l'inertie d'un groupe.
- **Pour chaque partition $\mathcal{P}_k$ de la hiérarchie en k classes ($k = 1, \ldots, n$), faire :**
  - Calculer l'inertie intra-classe $W(\mathcal{P}_k)$, telle que

$$W(\mathcal{P}_k) = \sum_{j=1}^{k} I(C_j) \quad \text{où } \mathcal{P}_k = (C_1, \ldots, C_k) \, .$$

- **Fin de la boucle.**
- Tracer un graphe représentant l'inertie intra-classe $W(\mathcal{P}_k)$ en fonction du nombre de classes k.
- Chercher le coude le plus marqué, et ainsi fixer le nombre de classes k_e pour l'élagage.
- Supprimer les noeuds dont l'indice est plus grand que $2\,k_e - 1$.

ALGORITHME 3.4 : Algorithme d'élagage par l'inertie.

3.3 Le recollement des feuilles

L'étape d'élagage permet de regrouper des points qui n'auraient pas dû être séparés, mais seulement s'ils sont issus d'un même noeud. Comme des mauvaises coupures peuvent être nécessaires pour en obtenir de bonnes, on peut se demander si une partie des points séparés lors d'une mauvaise coupure ne devraient pas être regroupés. Prenons l'exemple de la figure 3.2, de la page 26. La première coupure sépare des points d'une même classe, mais elle est nécessaire pour ensuite pouvoir isoler les deux autres classes (coupures 2 et 3). Il faudrait donc pouvoir recoller les deux parties de la classe séparées au début, et cela n'est pas possible par l'élagage (si on ne coupe pas, on ne parvient pas à isoler les deux autres classes). Et c'est pourquoi on a besoin de cette nouvelle étape.

Jusqu'ici, la classification est toujours présentée sous forme d'un arbre, et on a donc toujours une hiérarchie de partitions emboîtées. Mais comme nous allons maintenant tenter de recoller toutes les feuilles deux à deux, nous allons perdre le caractère hiérarchique de la classification. C'est pourquoi ces méthodes sont des méthodes de partitionnement, et non des méthodes hiérarchiques.

Nous allons voir deux critères pour effectuer ce recollement. Il s'agit aussi du **Gap Test**, et de l'**inertie**. Remarquons que si on utilise le Gap Test pour l'élagage, c'est également ce critère qu'il faudra utiliser pour le recollement, et de même pour l'inertie.

3.3.1 Recollement par le Gap Test

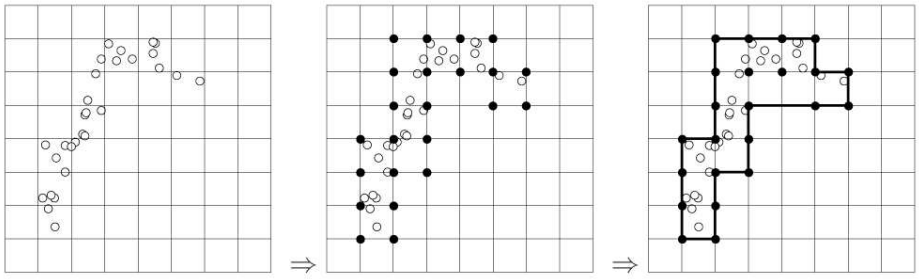
Considérons deux groupes C_1 et C_2 tels que, s'ils étaient recollés, ils formeraient un groupe connexe. Une première approche pour essayer de les recoller est d'effectuer le Gap Test selon la variable qui les a séparés au début. C'est l'approche **unidimensionnelle**. Ce test est bien différent de celui de l'élagage, car à l'étape d'élagage, on a considéré l'ensemble des points séparés par la coupure, alors qu'ici, on ne va considérer sur cette coupure qu'une partie des points (ceux des groupes C_1 et C_2).

Remarquons que le recollement est "associatif". En effet, si les groupes C_1 et C_2 doivent être recollés, ainsi que les groupes C_2 et C_3, alors ces trois groupes ne doivent plus en former qu'un. C'est ainsi qu'on obtient l'algorithme 3.5.

Néamoins, il s'est avéré que ce test ne donnait pas de résultats très satisfaisants, et c'est pourquoi [Pirçon, 2004] s'est penché sur un Gap Test **multidimensionnel**. Comme l'enveloppe convexe surestime les groupes, spécialement quand ils ne sont pas convexes, il s'est tourné vers l'**enveloppe convexe faible** ([Schmitt and Mattioli, 1993, page 164]). Rappelons que dans le cas du processus de Poisson non homogène, avant de calculer l'enveloppe convexe faible, il faut effectuer le changement de variables détaillé dans la section 2.1.2.

Pour établir l'enveloppe convexe faible, une trame carrée 4-connexité est utilisée. L'enveloppe convexe faible est donc l'ensemble des hypercarrés contenant des observations. Ce principe est illustré par la figure 3.5. En notant $\overline{C}$ l'enveloppe convexe faible du groupe C, la mesure du trou Δ est donnée par

$$m(\Delta) = m\big(\overline{C_1 \cup C_2}\big) - m\big(\overline{C_1}\big) - m\big(\overline{C_2}\big).$$



(a) Placement des données dans la trame

(b) Mise en évidence des points de la trame dont un des carrés contient au moins une donnée

(c) Calcul de l'enveloppe convexe

FIG. 3.5 – Principe de construction de l'enveloppe convexe faible, basée sur une trame carrée 4-connexité ([Pirçon, 2004]).

On peut alors appliquer la formule du Gap Test multidimensionnel :

$$\frac{n\, m(\Delta)}{m\left(\overline{C^1 \cup C^2}\right)} - \log n - (p-1)\log(\log n) - \log \kappa \geq -\log\big(-\log(1-\alpha)\big); \quad (3.3.1)$$

avec $n = \#(C_1 \cup C_2)$, p le nombre de variables, $\kappa = 1$ (forme convexe rectangulaire), et α est le niveau du test.

On a donc que, si l'équation 3.3.1 est vérifiée, le Gap Test rejette l'hypothèse nulle (un seul groupe), et donc il ne faut pas recoller C_1 et C_2. Sinon, ils sont recollés, et il faut alors vérifier si ce nouveau groupe $(C_1 \cup C_2)$ ne doit pas être recollé à chacun des autres groupes... L'algorithme 3.6 détaille le fonctionnement du Gap Test multidimensionnel.

Voici les différents algorithmes de recollement par le Gap Test.

Soit $\alpha = 0.05$, le niveau du test.
On suppose l'arbre de classification construit et élagué.
- **Pour chaque couple de feuilles de père différent, faire :**
  - Soient C^1 et C^2, les deux feuilles considérées (C^1 contenant n_1 points, et C^2, n_2 points, avec $n_1 + n_2 = n$). Soit D le domaine complet contenant les n points.
  - Retrouver la variable var_{max} qui a séparé ces deux ensembles de points.
  - Si le processus de Poisson est non homogène :
    - Estimer l'intensité $\lambda(t)$ des n points, selon la variable var_{max} ;
    - Transformer les données par le changement de variables vu en 2.1.2 (page 16) pour qu'elles soient des réalisations d'un processus de Poisson homogène :
$$\tau(x) = \int_0^x \lambda(t)\, dt\ .$$
  - Calculer la place de la statistique d'ordre du trou Δ entre C^1 et C^2 parmi l'ensemble des trous entre les points : i tel que $\Delta = S_{(n-i)}$.
  - Si $\dfrac{n\, m(\Delta)}{m(D)} - \log n \leq c$ avec $\displaystyle\sum_{j=0}^{i} \frac{(e^{-c})^j}{j!} e^{-e^{-c}} = 1 - \alpha$,
    alors effectuer le recollement de C^1 avec C^2 ;
  - Sinon, ne pas recoller C^1 et C^2.
- **Fin de la boucle.**
- Mettre les groupes à jour.

ALGORITHME 3.5 : Algorithme du recollement par le Gap test unidimensionnel.

Soit $\alpha = 0.05$, le niveau du test.
On suppose l'arbre de classification construit et élagué.
- **Pour chaque couple de feuilles de père différent, faire :**
  - Soient C^1 et C^2, les deux feuilles considérées (C^1 contenant n_1 points, et C^2, n_2 points, avec $n_1 + n_2 = n$).
  - Si le processus de Poisson est non homogène :
    Pour chaque variable ($j = 1, \ldots, p$), faire :
    - Estimer l'intensité $\lambda_j(t)$ des n points selon la variable var_j ;
    - Transformer les données par le changement de variables vu en 2.1.2 (page 16) pour qu'elles soient des réalisations d'un processus de Poisson homogène :
    $$\tau_j(x_j) = \int_0^{x_j} \lambda_j(t)\, dt \ .$$
    Fin de la boucle.
  - Positionner les données (transformées, dans le cas non homogène) dans la trame.
  - Mettre en évidence les points de la trame dont un des carrés contient au moins une donnée.
  - Estimer les enveloppes convexes faibles de C^1, de C^2 et de $C^1 \cup C^2$, et calculer leurs mesures respectives :
  $$m(\Delta) = m\left(\overline{C^1 \cup C^2}\right) - m\left(\overline{C^1}\right) - m\left(\overline{C^2}\right).$$
  - Si $\dfrac{n\, m(\Delta)}{m\left(\overline{C^1 \cup C^2}\right)} - \log n - (p-1)\log(\log n) \leq -\log\big(-\log(1-\alpha)\big)$,
    alors recoller C^1 et C^2, et inclure le nouveau groupe formé à l'ensemble des groupes sur lesquels il faut tester le recollement ;
  - Sinon, les groupes restent séparés.
- **Fin de la boucle.**
- Mettre à jour les groupes.

ALGORITHME 3.6 : Algorithme du recollement par le Gap test multidimensionnel.

3.3.2 Recollement par l'inertie

Le recollement par inertie fonctionne de la même façon que l'élagage par l'inertie. On commence par la partition avec k_ϵ classes, fournie par l'élagage. On va recoller deux classes, choisies de manière à faire augmenter au maximum l'intertie intra-classe. De même, avec notre partition à $k_\epsilon - 1$ classes, on va rechercher la meilleure partition en $k_\epsilon - 2$ classes, et ainsi de suite... Une fois qu'on a la meilleure partition en $2, 3, ..., k_\epsilon$ classes, on utilise la méthode du coude (tout comme pour l'élagage) pour choisir le nombre optimal de classes.

Voici l'algorithme détaillant le fonctionnement de cette méthode :

On suppose l'arbre de classification construit et élagué, tel qu'il contienne k_e classes. Soit $I(\cdot)$ l'inertie d'un groupe.

- **Pour tout** $k = k_e, \ldots, 2$**, faire :**
  - Notons $\mathcal{P}_k = \{C_1, \ldots, C_k\}$ la partition en k classes.
  - Trouver le meilleur recollement pour passer de k à $k-1$ classes (celui qui maximise l'inertie intra-classe) :

$$W_{max}(\mathcal{P}_{k-1}) = \max_{\Omega} W\Big(C_1, ..., C_{i-1}, C_i \cup C_j, C_{i+1}, ..., C_{j-1}, C_{j+1}, ..., C_k\Big)$$

$$(i_{max}, j_{max}) = \arg\max_{\Omega} W\Big(C_1, .., C_{i-1}, C_i \cup C_j, C_{i+1}, .., C_{j-1}, C_{j+1}, .., C_k\Big)$$

$$\text{où } \Omega = \Big\{C_i, C_j \in \mathcal{P}_k \,\Big|\, C_i \text{ et } C_j \text{ de père différent}\Big\}.$$

$$\rightarrow \mathcal{P}_{k-1} = (C_1, \ldots, C_{i-1}, C_i \cup C_j, C_{i+1}, \ldots, C_{j-1}, C_{j+1}, \ldots, C_k)$$

- **Fin de boucle.**
- Tracer un graphe représentant l'inertie intra-classe $W_{max}(\mathcal{P}_k)$ en fonction du nombre de classes k.
- Chercher le coude le plus marqué, et ainsi fixer le nombre de classes k_r pour le recollement.
- Recoller tous les groupes nécessaires à l'obtention de la partition $\mathcal{P}_{k_r}$ de la boucle ci-dessus.

ALGORITHME 3.7 : Algorithme du recollement par l'inertie.

3.4 Les différentes méthodes

Maintenant que nous avons vu le fonctionnement des trois étapes de classifications des méthodes, voyons quelque peu leur fonctionnement général. Voici tout d'abord l'algorithme général de fonctionnement des cinq méthodes :

- **Choix de la méthode :**
  - Hypothèse sur les données : *HOPP* ou *NHOPP* ?
  - Si *NHOPP* :
    Estimation de l'intensité ?
    - soit *SONHOPPHI* ;
    - soit *SONHOPPKI* ;
    - soit *UNHOPPHI* ;
    - soit *UNHOPPKI* ;
  - Elagage et recollement ?
    Gap test unidimensionnel, Gap test multidimensionnel ou inertie ?
  - Si *NHOPP* et Gap test :
    Estimer l'intensité du processus par les histogrammes pour la coupure, mais utiliser l'estimation par les noyaux pour l'élagage et le recollement ?
- **Construction de l'arbre :** Algorithme 3.1
- **Elagage de l'arbre :**
  - Si Gap test (unidimensionnel ou multidimensionnel) : Algorithme 3.3
  - Si inertie : Algorithme 3.4
- **Recollement de l'arbre :**
  - Si Gap test unidimensionnel : Algorithme 3.5
  - Si Gap test multidimensionnel : Algorithme 3.6
  - Si inertie : Algorithme 3.7

ALGORITHME 3.8 : Algorithme général des nouvelles méthodes.

Remarquons que la méthode *UNHOPPKI* a déjà été adaptée pour traiter des objets symboliques de type intervalle (une version dépouillée de recollement de cette méthode est incluse dans le logiciel [SODAS, 2], sous le nom de *SCLASS*), et le but de ce présent mémoire est d'adapter la méthode *HOPP* aux données symboliques de type intervalle.

Comparons quelque peu ces différentes méthodes. D'un point de vue algorithmique, la construction de l'arbre se fait en $\mathcal{O}\big(n\,(log\,n)^2\big)$ pour les méthodes *HOPP*, et *NHOPPHI*. Les méthodes faisant appel à l'estimation par les noyaux sont en $\mathcal{O}\big(n^2\big)$. Il en est de même pour l'élagage par le Gap Test, et pour le recollement par le Gap Test unidimensionnel. Le recollement par le Gap Test multidimensionnel se fait, lui , en $\mathcal{O}\big(n\,log\,n\big)$ pour les méthodes *HOPP* ou *NHOPPHI* , et toujours en $\mathcal{O}\big(n^2\big)$ pour les méthodes *NHOPPKI*. Les méthodes basées sur l'estimation de l'intensité d'un processus de Poisson non homogène par les noyaux est donc beaucoup plus coûteuse en temps que les trois autres méthodes.

Si l'on compare les résultats que fournissent ces différentes méthodes, on peut tirer plusieurs conclusions. Ainsi, le recollement par le Gap Test multidimensionnel est bien meilleur que l'unidimensionnel. En effet, des classes peuvent être proches selon une variable, alors que toutes leurs autres variables diffèrent de beaucoup. Dans ce cas, le Gap Test unidimensionnel recollera les deux classes, alors que le multidimensionnel ne le fera pas.

Une autre observation qui a été faite est que l'élagage et le recollement par le Gap Test donnent de meilleurs résultats quand on estime l'intensité du processus par les noyaux plutôt qu'avec les histogrammes. Cependant, la construction de l'arbre en estimant l'intensité du processus par les noyaux est à éviter, car elle met beaucoup plus de temps pour fournir des résulats similaires.

Concernant l'élagage et le recollement par l'inertie, ceux-ci sont déconseillés à cause de leur complexité algorithmique. De plus, ces méthodes recherchent des classes hypersphériques, alors que le Gap Test permet de retrouver n'importe quel type de classe.

Ainsi, les méthodes qui se détachent le plus sont donc *HOPP*, *SONHOPPHI* et *UNHOPPHI*, en prenant le Gap Test multidimensionnel (de préférence avec l'estimation de l'intensité par les noyaux). Dans sa thèse, [Pirçon, 2004] avait surtout mis en avant la méthode *SONHOPPHI*, avec Gap Test multidimensionnel (estimation par les noyaux). Nous voudrions maintenant savoir si la méthode *HOPP*, qui est bien plus simple puisqu'il ne faut pas estimer d'intensité, est capable de fournir des résultats similaires, dans le cas des données symboliques de type intervalle. Nous allons donc nous pencher maintenant sur ce type de données particulières.

Chapitre 4

Les données symboliques

Jusqu'à présent, nous avons toujours considéré des données de type "classique". Voyons maintenant ce que sont les données symboliques, et ce qui les différencie des données classiques.

4.1 Données classiques

Les variables classiques se partagent en deux catégories :

1. **les variables quantitatives**, dont le domaine d'observation est réel. On y distingue les variables continues (un nombre infini non dénombrable de valeurs, comme par exemple un chiffre d'affaires) et les variables discrètes (nombre fini ou infini dénombrable, comme le nombre d'employés d'une entreprise).
2. **les variables qualitatives, ou catégoriques**. Il s'agit de variables dont le domaine est fini, et dont les valeurs ne se prêtent pas à des calculs numériques. On les départage en variables ordinales si leurs valeurs peuvent être hiérarchisées (par exemple une appréciation : {mauvais, moyen, bon, très bon}) , et en variables nominales s'il n'y a aucun ordre logique entre leurs valeurs (par exemple des nationalités, ou des noms d'entreprises). Un cas particulier de variables nominales est les variables dites binaires, ou dichotomiques, dont le domaine ne contient que deux valeurs (comme le sexe d'une personne).

Remarquons que nous avons toujours, jusqu'ici, utilisé implicitement des variables quantitatives, bien que certaines méthodes vues précédemment s'adaptent aux autres types de variable. Penchons-nous maintenant sur ces variables "symboliques"...

4.2 Données symboliques

Les données symboliques se partagent elles aussi en différents groupes : il y a, premièrement, les variables multivaluées. Une variable Y est multivaluée si les valeurs $Y(x_k)$, avec x_k les objets que l'on étudie, sont des sous-ensembles finis du domaine de Y. On peut alors avoir des variables multivaluées catégoriques, dont les valeurs sont un nombre fini de catégories, et des variables multivaluées quantitatives, dont les valeurs sont un ensemble fini de nombres réels.

Il y a également les variables de type intervalle, dont les valeurs $Y(x_k)$ sont des intervalles (fermés bornés) de $\mathbb{R}$.

Il y a, enfin, les variables modales, dont chaque valeur $Y(x_k)$ est une variable (valeur, intervalle, etc.), associée à une fréquence. Par exemple, si on étudie le prix d'une voiture, on peut retourner chaque niveau de finition (ou sa fourchette de prix), avec son pourcentage de vente. On aurait alors quelque chose du genre :

$$Y(x_k) = \big\{(\text{"basic"}\,,\,0.3)\,;\,(\text{"business"}\,,\,0.4)\,;\,(\text{"luxury"}\,,\,0.2)\,;\,(\text{"sport"}\,,\,0.1)\big\}.$$

On voit donc qu'il existe, pour toute variable modale, un diagramme en bâtons (ou un histogramme) qui lui est associé.

On peut facilement obtenir des données symboliques à partir de données classiques. En effet, au lieu de considérer les nombreux objets x_k que l'on veut étudier, on les répartit en classes C_i, et on construit les variables symboliques $\overline{Y_j}$ à partir des variables Y_j telles que

$$\overline{Y_j}(C_i) \text{ caractérise l'ensemble } \big\{Y_j(x_k)\,|\,x_k \in C_i\big\}.$$

Un grand intérêt du passage aux données symboliques est donc la rédution du nombre d'objets à étudier : on étudie des classes d'individus, appelées **objets du second ordre**, et non les individus eux-mêmes (**objets du premier ordre**).

On peut ainsi transformer des variables quantitatives en variables multivaluées quantitatives ou en intervalles, des variables catégoriques en variables multivaluées catégoriques. On peut également avoir des variables modales, en associant à chaque catégorie sa fréquence ...

4.3 Dissimilarité entre objets symboliques

Avec des objets classiques, il est relativement aisé de mesurer des "distances" entre objets. Pour des variables quantitatives, il existe bon nombre de mesures de distance, telle la distance euclidienne, pour ne citer qu'elle ; et pour des variables qualitatives, on peut retourner le nombre de catégories séparant les objets si les variables sont ordinales, ou si elles sont nominales, retourner 1 ou 0 selon que les objets aient, ou non, la même valeur de la variable.

Variables de type intervalle

Dans le cas des données symboliques, des mesures de distances doivent être adaptées. Ainsi, on peut définir la distance entre deux intervalles $[\alpha_1, \beta_1]$ et $[\alpha_2, \beta_2]$ par la

1. **distance de Hausdorff** :
$$d\big([\alpha_1, \beta_1]\,,\, [\alpha_2, \beta_2]\big) \;=\; \max\big\{|\alpha_2 - \alpha_1|\,,\, |\beta_2 - \beta_1|\big\};$$
2. **distance** L_1 :
$$d\big([\alpha_1, \beta_1]\,,\, [\alpha_2, \beta_2]\big) \;=\; |\alpha_2 - \alpha_1| \,+\, |\beta_2 - \beta_1|;$$
3. **distance** L_2 :
$$d\big([\alpha_1, \beta_1]\,,\, [\alpha_2, \beta_2]\big) \;=\; (\alpha_2 - \alpha_1)^2 \,+\, (\beta_2 - \beta_1)^2.$$

Variables multivaluées

Pour des variables multivaluées, on peut associer à chaque objet x_k la fréquence de chaque catégorie c_s de la variable étudiée (Y). Cette fréquence se note $q_{x_k}(c_s)$, et vaut

$$q_{x_k}(c_s) \;=\; \left\{ \begin{array}{cl} \frac{1}{|Y(x_k)|} & \text{si } c_s \in Y(x_k) \\ 0 & \text{sinon.} \end{array} \right.$$

On transforme donc les données de la façon suivante :

$$\left(\begin{array}{cc} \underline{x_k} & \underline{\textit{Couleurs}} \\ \\ x_1 & \{C\} \\ x_2 & \{A, C, D\} \\ x_3 & \{A, B\} \end{array} \right) \Longrightarrow \left(\begin{array}{ccccc} \underline{x_k} & \underline{A} & \underline{B} & \underline{C} & \underline{D} \\ \\ x_1 & 0 & 0 & 1 & 0 \\ x_2 & \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} \\ x_3 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \end{array} \right)$$

On peut alors définir des distances :

- **Distance** L_1 :

$$d(x_a, x_b) = \sum_s \left| q_{x_a}(c_s) - q_{x_b}(c_s) \right|;$$

- **Distance** L_2 :

$$d(x_a, x_b) = \sum_s \left(q_{x_a}(c_s) - q_{x_b}(c_s) \right)^2;$$

- **Distance de de Carvalho** :

$$d(x_a, x_b) = \sum_s \left(\gamma\, q_{x_a}(c_s) - \delta\, q_{x_b}(c_s) \right),$$

$$avec\ \gamma = \begin{cases} 1 & \text{si } c_s \in Y(x_a) \text{ et } c_s \notin Y(x_b), \\ 0 & \text{sinon.} \end{cases}$$

$$\text{et } \delta = \begin{cases} 1 & \text{si } c_s \notin Y(x_a) \text{ et } c_s \in Y(x_b), \\ 0 & \text{sinon.} \end{cases}$$

Variables modales

Enfin, pour ce qui est des **variables modales**, il suffit de les considérer comme des variables multivaluées, et de prendre pour fréquence $q_{x_k}(c_s)$ les fréquences associées aux différents modes.

Données multivariées

Remarquons pour terminer ce chapitre que pour mesurer la distance entre des objets caractérisés par plusieurs variable Y_j $(i = 1, ..., p)$, on utilise la distance d telle que

$$d(x_a, x_b) = \left(\sum_{j=1}^{p} d_j^2(x_{a,j}\ ,\ x_{b,j}) \right)^{\frac{1}{2}},$$

où chaque d_j est la mesure de distance choisie pour la variable j.

Chapitre 5

Méthodes de classification symbolique

Nous allons présenter dans ce chapitre différentes méthodes de classification adaptées aux données de type intervalle. Dans les chapitres ultérieurs, nous détaillerons la nouvelle méthode *SHOPP*, puis nous la comparerons à ces méthodes. Commençons par *SCLASS*, nous verrons ensuite *DIV* et *SCLUST*.

5.1 La méthode SCLASS

La méthode *SCLASS* est l'extension symbolique de la méthode *UNHOPPKI* présentée au chapitre 3. Cette méthode est due à [Rasson et al., 2005], et est disponible dans le logiciel [SODAS, 2]. Remarquons que la version de *SCLASS* disponible actuellement n'inclu pas l'étape de recollement. *SCLASS* est donc hiérarchique (monothétique divisive), et fourni un arbre de décision.

SCLASS modélise les intervalles par leurs coordonnées *(milieu, longueur)*, comme l'illustre la figure 5.1. Pour chaque variable I_i de type intervalle, la méthode va en fait considérer deux variables M_i et L_i, qui sont respectivement le milieu et la longueur de l'intervalle I_i. Cependant, les coupures ne vont se faire que selon les centres des intervalles.

Plus précisément, *SCLASS* considère toutes les partitions en deux classes obtenues par une coupure parallèle à l'axe des longueurs, selon l'ordre des centres des intervalles, et elle va rechercher l'intervalle $]M_i\,,\,M_{i+1}[$ tel que

$$\int_{M_i}^{M_{i+1}} q_M(x)\,dm(x) \;+\; \int_{L_i}^{L_{i+1}} q_L(y)\,dm(y)$$

soit minimal, avec q_M et q_L les intensités intégrées respectivement sur l'axe des milieux, et des longueurs. La méthode retourne alors la partition obtenue en effectuant la coupure dans l'intervalle $]M_i\,,\,M_{i+1}[$.

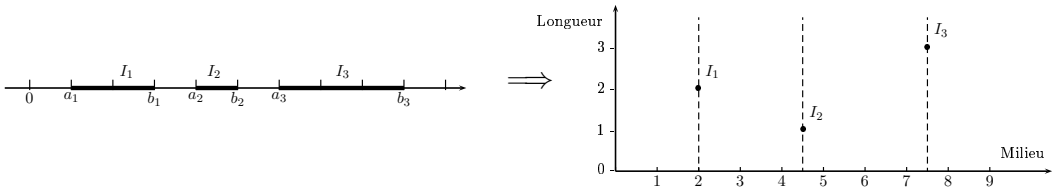


FIG. 5.1 – Transformation de la représenration des données intervalles

Comme dans le cas des données traditionnelles, les noeuds sont donc toujours divisés en deux sur base de questions binaires, mais portant uniquement sur les variables M_i. Le critère d'arrêt est un nombre minimal de points par noeud, et l'élagage se fait toujours par le Gap Test.

5.2 La méthode DIV

Nous devons cette méthode à [Chavent, 1997]. Il s'agit d'une méthode hiérarchique monothétique divisive, dont le critère est basé sur l'inertie. Elle traite aussi bien des variables quantitatives (classiques et intervalles) que des variables ordinales (classiques, multivaluées et modales). Nous ne présenterons ici que le cas des données de type intervalle.

Cette méthode recherche à minimiser l'inertie intra-classe W d'une partition en k classes C_l telle que

$$W = \sum_{l=1}^{k} I_{g_{(l)}}(C_l) = \sum_{l=1}^{k} \sum_{x_i \in C_l} d^2(x_i, g_{(l)}) = \sum_{l=1}^{k} \frac{1}{2\,n_l} \sum_{x_i \in C_l} \sum_{x_j \in C_l} d^2(x_i, x_j)$$

où on prend pour d la distance de Hausdorff, et où $g_{(l)}$ représente le centre de gravité de la l ième classe.

Tout comme dans les méthodes divisives déjà vues, pour pallier à la complexité algorithmique, *DIV* va rechercher la meilleure bipartition d'une classe induite par des questions binaires. Ainsi, pour couper une classe C en deux sous-classes C_1 et C_2, il va calculer $\Delta C = I(C) - I(C_1) - I(C_2)$ pour chaque valeur de coupure (entre deux milieux d'intervalles consécutifs), puis il effectuera la coupure à l'endroit où il avait obtenu un ΔC maximal.

L'inertie $I(.)$ se calcule par $I(C) = \frac{1}{2\ \#(C)} \sum_{x_i \in C} \sum_{x_j \in C} d^2(x_i, x_j)$.

Le choix de la classe à couper se fait en calculant pour chaque classe le ΔC, et en prenant alors la classe qui apporte la plus grande diminution de l'inertie. Le critère d'arrêt est un nombre maximal L de divisions (fixé au préalable par l'utilisateur), et la méthode retourne alors une hiérarchie de partitions dont les groupes de base sont les $L + 1$ classes obtenues après la dernière division, et qui est indexée, pour chaque classe C_i, par la valeur ΔC_i.

Remarquons que cette hiérarchie retournée par *DIV* peut s'utiliser comme un arbre de décision, puisque les coupures se font par des questions binaires. Et donc, pour traiter un grand nombre d'individus, on peut se contenter d'établir les groupes sur base d'un échantillon représentatif, et de "classer" les autres individus dans les bonnes classes avec la hiérarchie obtenue.

Signalons aussi que la méthode *DIV* est disponible dans [SODAS, 2].

5.3 La méthode SCLUST

Cette méthode est complètement différente de toutes celles déjà présentées puisqu'il s'agit d'une pure méthode de partitionnement, qui cherche à améliorer une partition (au sens d'un critère W), sans en modifier le nombre de groupes.

La méthode commence par initialiser une partition aléatoire en k classes $P = \{C_1, ..., C_k\}$, avec k fixé par l'utilisateur.

Les itérations se font en deux parties : tout d'abord, la méthode associe à chaque classe "un représentant", appelé prototype. C'est l'**étape de représentation**. Dans les cas des données intervalles, le prototype d'une classe C est l'hyperrectangle de gravité L tel que

$$L = \left(\left[\frac{1}{n} \sum_{x_i \in C} \alpha_{i1} \, , \, \frac{1}{n} \sum_{x_i \in C} \beta_{i1} \right] , \, ... \, , \, \left[\frac{1}{n} \sum_{x_i \in C} \alpha_{ip} \, , \, \frac{1}{n} \sum_{x_i \in C} \beta_{ip} \right] \right).$$

Ensuite, *SCLUST* réaffecte chaque individu à la classe dont il est le plus proche du prototype ; c'est l'**étape d'affectation**.

On cherche donc la partition en k classes $P^* = \{C_1^*, ..., C_k^*\}$ et le vecteur de prototypes L_k^* représentant les classes de P^* qui minimisent

$$W(P, L) = \sum_{l=1}^{k} \sum_{x_i \in C_l} d(x_i, L^{(l)})$$

en prenant pour distance d (entre intervalles) la distance de Hausdorff.

On peut montrer que sous certaines conditions de régularité, il y a convergence de l'algorithme. Notons aussi que cette méthode *SCLUST* est disponible dans le logiciel [SODAS, 2].

Chapitre 6

La nouvelle méthode *SHOPP*

Nous allons présenter dans ce chapitre notre nouvelle méthode de classification automatique traitant des données symboliques de type intervalle. Celle-ci porte le nom de *SHOPP*, pour *Symbolic Homogeneous Poisson Process*. C'est une adaptation aux données intervalles de la méthode classique *HOPP* présentée au chapitre 3, de la même manière que la méthode *SCLASS* est issue de la méthode *UNHOPPKI*, également présentée dans ce chapitre 3.

La modélisation des intervalles est la même que celle de *SCLASS* : on remplace chaque variable de type intervalle par deux variables simples, représentant pour une le centre des intervalles, et pour l'autre leur demi-longueur.

Dans notre travail d'adaptation, nous avons choisi de ne pas implémenter l'étape de recollement proposée par [Pirçon, 2004]. En effet, celle-ci nous semble fort gourmande en ressources informatiques. Cette étape a pourtant son utilité, et c'est pourquoi il nous semblerait intéressant de développer d'autres méthodes de recollement plus efficaces. Remarquons que la version de *SCLASS* disponible dans le logiciel [SODAS, 2] est, elle aussi, dépourvue de cette étape. Notre méthode *SHOPP* est donc une méthode hiérarchique divisive monothétique.

Celle-ci se compose de deux parties, à savoir une première étape de coupure, où la méthode crée un arbre de classification selon les coupures qu'elle effectue, et une seconde qui permet de simplifier l'arbre en supprimant les branches "inutiles".

Voyons ces deux étapes plus en détail.

6.1 Création de l'arbre

Tout comme *SCLASS*, les coupures se font en considérant toutes les partitions en 2 classes du groupe à couper qui sont obtenues en effectuant des coupures parallèles aux axes des milieux d'intervalle. La coupure retenue est celle qui maximise le trou entre les classes.

Plus formellement, on a que, pour chaque variable, *SHOPP* va rechercher l'intervalle $]M_i, M_{i+1}[$ qui maximise $m(\Delta)$, tel que

$$m(\Delta) \;=\; \big|M_i - M_{i+1}\big| + \big|L_i - L_{i+1}\big|,$$

où les M_i représentent la variable centre d'une variable intervalle, et les L_i sa variable demi-longueur. La coupure va ensuite s'effectuer au centre de cet intervalle $]M_i, M_{i+1}[$.

D'un point de vue pratique, pour une variable intervalle donnée, on va trier la base selon la colonne des centres de cette variable (les M_i), et on va ensuite mesurer le trou entre chaque individu et son suivant. On remarque que les longueurs d'intervalles n'interviennent que dans la mesure du trou Δ, et donc uniquement dans le choix de l'endroit où on va effectuer la coupure.

Pour effectuer une coupure, la méthode va rechercher pour chaque variable intervalle la mesure du plus grand trou que l'on peut obtenir, et effectuer sa coupure selon la variable qui propose le plus grand trou, entre les deux individus où ce trou est obtenu.

Les coupures cessent quand il y a moins d'un certain nombre d'individus dans la classe. Ce nombre à été fixé arbitrairement à 2 ; nous voulons effectivement toutes les coupures possibles, l'étape suivante se chargeant de supprimer celles qu'il ne fallait pas faire. Il y a cependant moyen d'ajouter un test d'arrêt supplémentaire, basé sur le test de Fisher (cf. 3.1.2, page 23). Celui-ci va tester si les deux classes obtenues après la coupure font en réalité partie de la même classe, ou si la coupure est justifiée. Il permet de savoir si une coupure est statistiquement significative ou non.

6.2 Elagage de l'arbre

L'élagage de l'arbre va se faire selon le Gap Test, tout comme pour *SCLASS*, à une différence près : le Gap Test n'étant valide que pour des données suivant un processus de Poisson **homogène**, *SCLASS* (basée sur l'hypothèse que les données suivent un processus de poisson **non homogène**) doit effectuer un changement de variable (celui-ci est présenté à la section 2.1.2, page 16). Comme *SHOPP* vérifie déjà cette hypothèse, on peut ici effectuer directement le test.

On va donc tester, pour chaque noeud contenant n points, si la coupure est bonne ou non, en testant l'hypothèse H_0 contre l'hypothèse alternative H_1, telles que :
- H_0 : les n points du noeud forment un seul domaine D,
- H_1 : les n points sont répartis dans deux domaines D_1, D_2 distincts.

Rappellons que le Gap Test a été présenté plus en détails dans la section 2.4, à la page 19.

Une fois toutes les coupures déclarées bonnes ou mauvaises, *SHOPP* supprime tous les bouts de branche qui ne contiennent que de mauvaises coupures, tout comme *SCLASS* et les méthodes de [Pirçon, 2004] présentées au chapitre 3.

Remarquons que, dans sa généralisation de la méthode *HOPP*, *SHOPP* peut toujours traiter des données quantitatives classiques. En effet, si on utilise des variables intervalle dont la longueur est toujours nulle, la méthode va fonctionner sans aucun problème, la partie de l'aire due aux longueurs d'intervalles sera systématiquement nulle, et on obtiendra exactement les mêmes résultats qu'avec la méthode traditionnelle *HOPP*.

Procédons maintenant aux tests de la nouvelle méthode *SHOPP*. Au cours du chapitre suivant, nous allons voir ce que donne notre nouvelle méthode sur cinq jeux de données à deux variables. Prendre des jeux de données bivariés permet en effet de visualiser clairement comment fonctionne la méthode. Nous verrons ensuite un jeu de données réelles, à huit variables.

Chapitre 7

Applications pratiques

Nous allons ici appliquer la nouvelle méthode *SHOPP* à différents jeux de données, et comparer ses résultats à ceux obtenus par les méthodes *SCLASS*, *DIV* et *SCLUST*. Nous utiliserons principalement des jeux de données artificiels à deux variables, de manière à pouvoir illustrer clairement les résultats, et nous terminerons par un jeu de données réelles plus ludique à huit variables.

7.1 Jeu de données *iBSP*

Commençons par une base assez simple : *iBSP*. Celle-ci se compose de 243 individus, répartis en trois groupes de taille 81. Ces groupes sont relativement bien séparés, et des coupures parallèles aux axes devraient permettre de les retrouver.

7.1.1 Par la méthode *SHOPP*

Dans sa version sans critère d'arrêt autre qu'un nombre minimal de points dans les classes, on se rend vite compte qu'une base de 243 individus sera découpée en un grand nombre de groupes par *SHOPP*. Et bien que l'élagage supprime toute une série de branches inutiles, cette méthode nous retourne quand même 56 classes ...

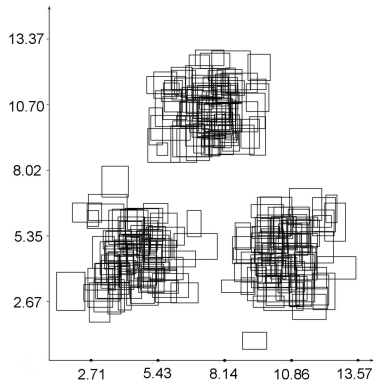
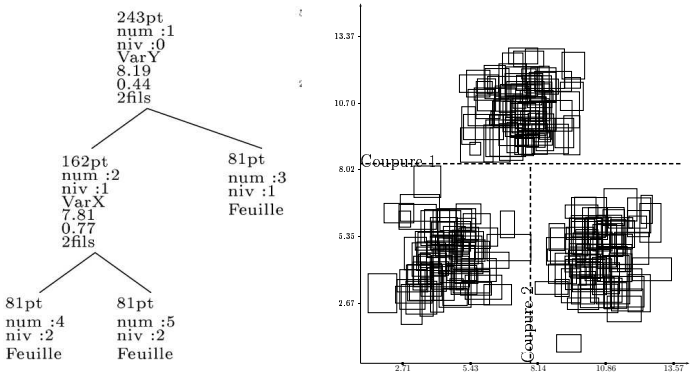


FIG. 7.1 – Présentation de la base *iBSP*.

Maintenant, dans sa version munie d'un test de Fisher comme critère d'arrêt supplémentaire, la méthode *SHOPP* nous fourni de bien meilleurs résultats ! En effet, celle-ci nous retourne les trois classes attendues, et ce sans avoir besoin de l'étape d'élagage. La figure 7.2 comprend l'arbre de classification, et un graphique illustrant les coupures. Chaque noeud de l'arbre précise le nombre d'individus qu'elle contient, son numéro, son niveau dans l'arbre puis, si le noeud n'est pas une feuille, la variable et les valeurs de coupure (milieu et demi-longueur).



(a) Arbre de classification

(b) Graphique montrant les coupures

FIG. 7.2 – Classification obtenue par *SHOPP*, muni du test de Fisher, pour la base de données *iBSP*.

7.1.2 Par la méthode *SCLASS*

La méthode *SCLASS*, basée sur le même principe, devrait nous fournir des résultats semblables à la méthode *SHOPP*. Il n'en est rien ! Comme on peut le constater sur l'arbre de classification ci-dessous (figure 7.3), *SCLASS* choisi pour commencer de séparer les deux groupes du bas par une coupure verticale, qui passe au travers du groupe du haut puis, à une exception près, continue de couper verticalement, au travers des classes...

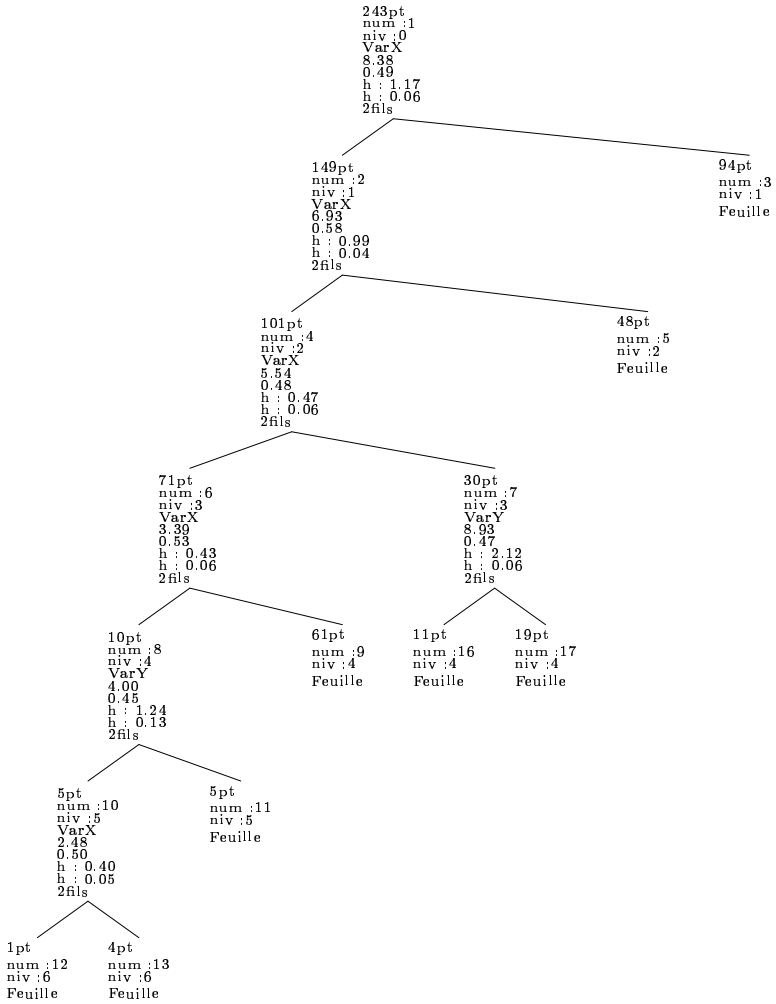


FIG. 7.3 – Arbre de classification obtenu par la méthode *SCLASS* pour la base de données *iBSP*.

Voici le graphique illustrant les coupures :

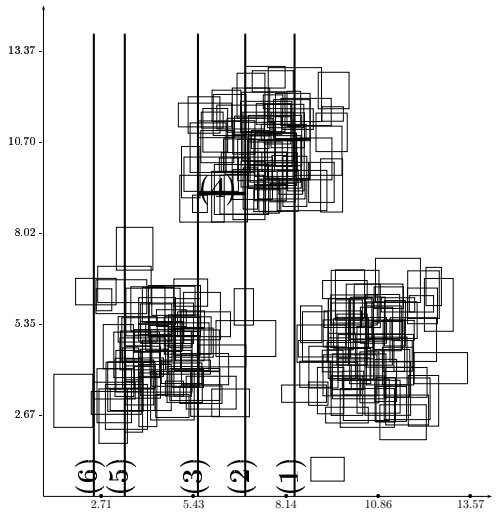


FIG. 7.4 – Résultats de la méthode *SCLASS*, pour la base de données *iBSP*.

Ce choix des coupures nous apparait peu compréhensible, surtout que la méthode *SHOPP*, qui lui est pourtant très proche, nous fourni des résultats bien différents... Voyons donc ce que nous propose la méthode *DIV*, qui est elle aussi divisive monothétique, mais basée sur un tout autre crièrte.

7.1.3 Par la méthode *DIV*

La méthode *DIV* nous fourni presque les résultats attendus. Celle-ci commence bien par isoler la classe du haut, puis sépare des classes du bas correctement. La seule différence avec la méthode *SHOPP* est la hauteur à laquelle la classe du haut est délimitée. En effet, on remarque que le rectangle dépassant du haut de la classe de gauche (l'individu n^o 35) est ici rattaché à la classe du haut et non à celle de gauche.

Remarquons que la méthode *DIV* demande comme paramètre le nombre de classes. Nous lui en avons demandé trois, ce qui peut expliquer que le rectangle isolé tout en bas n'ait pas été séparé de sa classe, lui, bien que l'individu n^o 35 l'ait été. Signalons que si on demande quatre classes, la quatrième est obtenue en coupant verticalement au milieu de la classe d'en bas à gauche (à l'abscisse 4.62).

Voici le graphique illustrant les coupures :

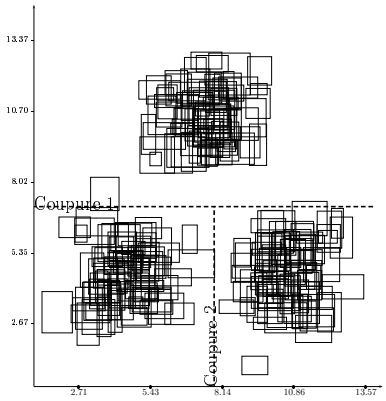


FIG. 7.5 – Résultats de la méthode *DIV*, pour la base de données *iBSP*.

7.1.4 Par la méthode *SCLUST*

La méthode *SCLUST* est une méthode fort différente des autres déjà testées, puisqu'elle n'est pas hiérarchique, et qu'elle travaille avec toutes les variables en même temps, en cherchant à améliorer un critère.
Cette méthode nous retourne ici les trois bonnes classes de 81 individus chacun. Voici un graphique incluant les prototypes des classes. Ceux-ci sont tracés en blanc sur le graphique :

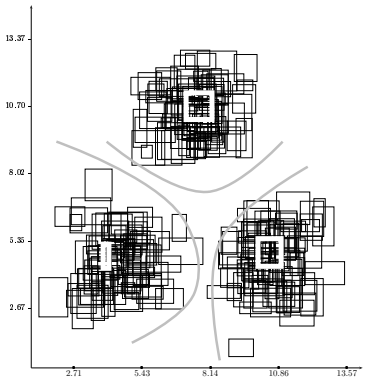


FIG. 7.6 – Résultats de la méthode *SCLUST*, pour la base de données *iBSP*.

7.2 Jeu de données *iGP*

Etudions maintenant la base *iGP*. Celle-ci est constituée de sept classes relativement bien séparées, et que l'on peut obtenir par des coupures parallèles aux axes. L'intérêt de cette base est qu'elle mèle des classes de faible effectif à des classes plus importantes, ainsi que des classes sphériques à des classes elliptiques. En voici une illustration :

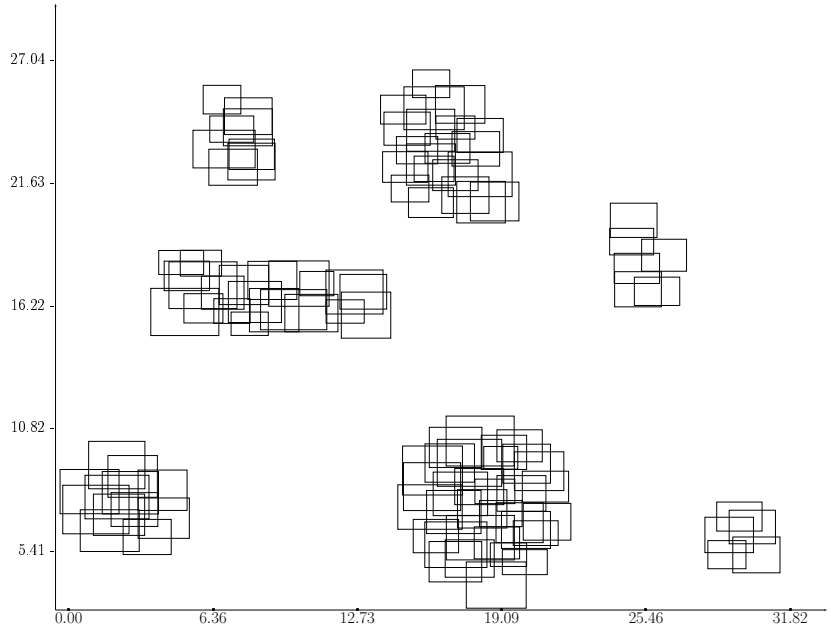
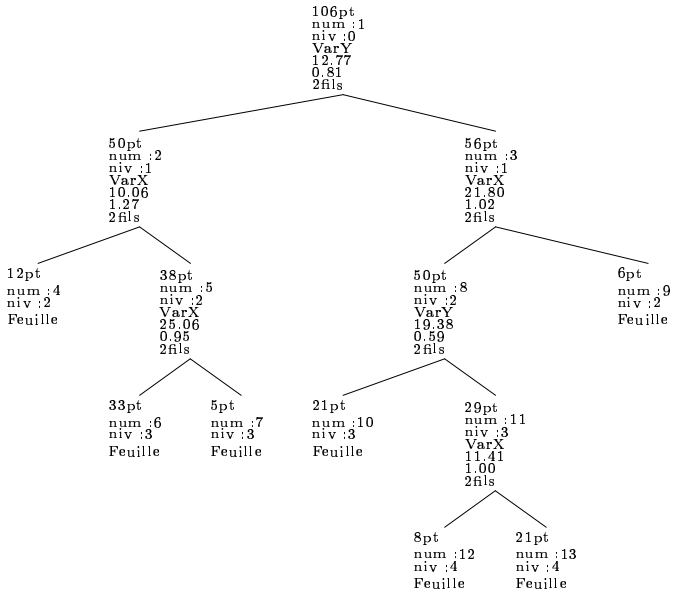


FIG. 7.7 – Présentation de la base *iGP*.

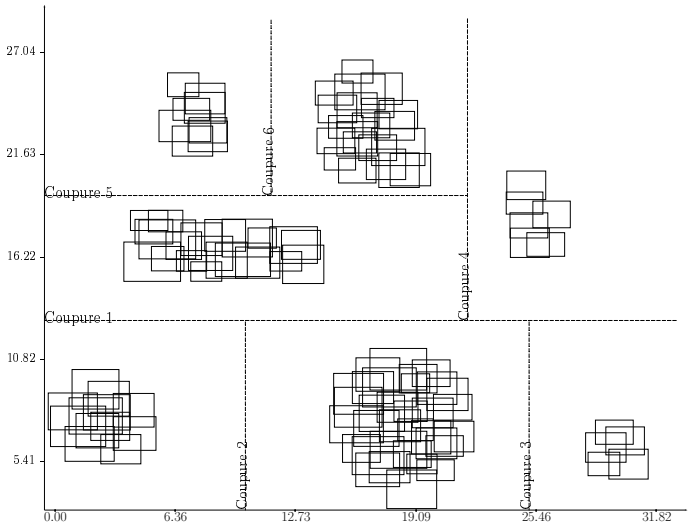
7.2.1 Par la méthode *SHOPP*

Dans sa version munie du test de Fisher, *SHOPP* parvient à retrouver les bons groupes avant même l'étape d'élagage, tandis que si on laisse les coupures se faire jusqu'au bout, l'élagage ne supprime qu'une partie des branches superflues. Il est donc préférable d'utiliser ce critère d'arrêt supplémentaire. Remarquons qu'il ne s'agit que d'un critère d'arrêt, et donc qu'avec ou sans ce test, les coupures se font aux mêmes endroits.

Voici l'arbre de classification retourné par la méthode *SHOPP*, munie du test de Fisher, ainsi que le graphique des données présentant les coupures :



(a) Arbre de classification



(b) Graphique illustrant les coupures

FIG. 7.8 – Classification obtenue par *SHOPP* pour la base *iGP*.

7.2.2 Par la méthode *SCLASS*

Nous n'avons pas pu obtenir de résultats complets de la part de *SCLASS* pour ce jeu de données, car après avoir effectué quelques coupures horizontales (une première correcte, puis les autres au travers des classes), le module SCLASS de [SODAS, 2] plante systématiquement. Signalons que ce module est encore jeune et n'a pas été complètement débuggé.

La figure ci-dessous nous montre les premières coupures de *SCLASS*. Remarquons que cette méthode crée son arbre de classification en profondeur et non en largeur, et comme il s'arrête avant la fin, toute la partie supérieure du graphique n'est pas traitée...

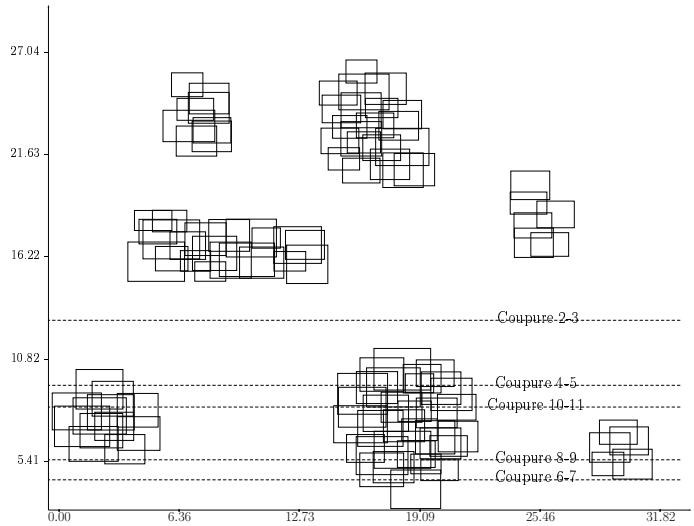


FIG. 7.9 – Classification (non complète) obtenue par *SCLASS* pour la base *iGP*.

Passons maintenant à une autre méthode hiérarchique monothétique divisive également inclue dans [SODAS, 2].

7.2.3 Par la méthode *DIV*

La méthode *DIV* nous fourni exactement les mêmes classes que la méthode *SHOPP*. Notons cependant qu'elle obtient les 4 classes de la partie supérieure par des coupures différentes de celles de *SHOPP*, alors que pour la partie inférieure, elle prend précisément les mêmes valeurs de coupure.

Voici la classification proposée par la méthode *DIV* :

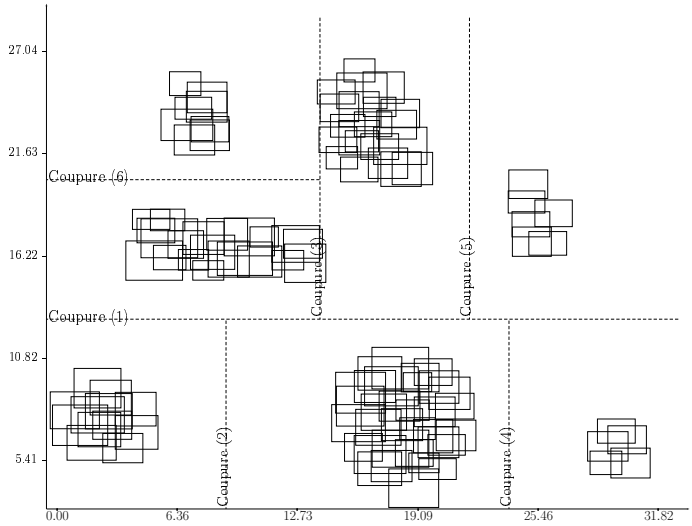
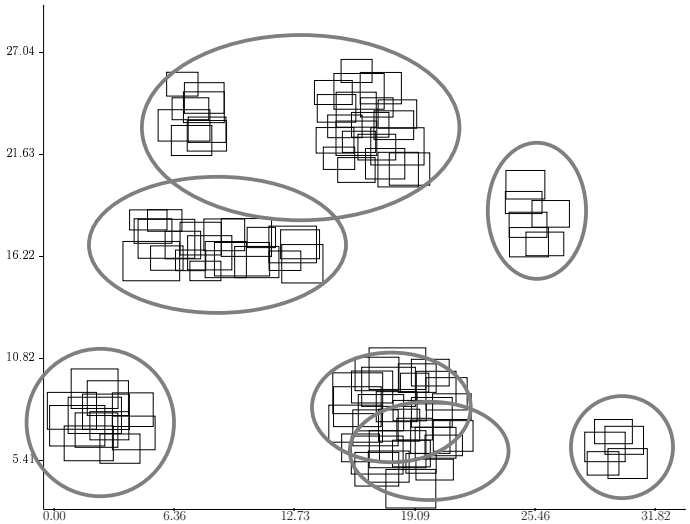


FIG. 7.10 – Classification obtenue par *DIV* pour la base *iGP*.

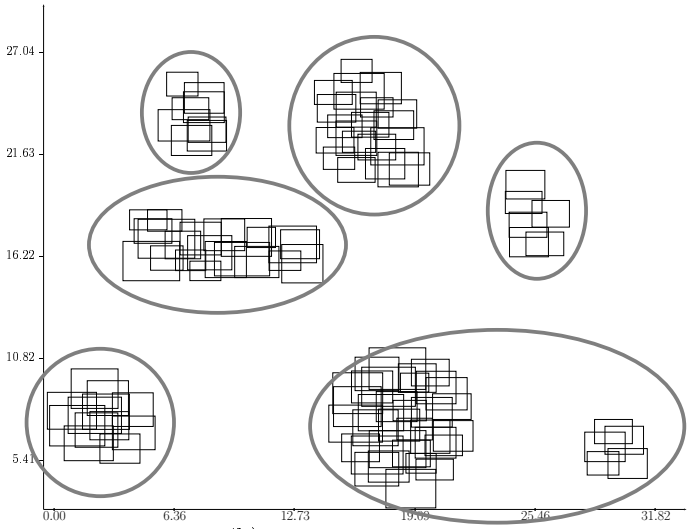
7.2.4 Par la méthode *SCLUST*

La méthode *SCLUST*, tout comme *DIV* dont nous venons d'analyser les résultats, demande comme paramètre le nombre de classes à retourner. Si nous lui en demandons 7, elle préfère couper la grosse classe du bas, plutôt que d'isoler les deux classes les plus hautes. Remarquons néamoins que si on ne demande que 6 classes, les deux classes du haut seront bien séparées, mais les deux classes d'en-bas à droite (celle coupée dans la partition en 7 groupes et celle à sa droite) seront fusionnées. Des figures illustrant ces découpages sont visibles ci-après.

Ceci termine la première étude de cette base. Reprenons-la maintenant avec les longueurs d'intervalle multipliées par un facteur 3 ; notre critère n'effectuant ses coupures que selon les centres des intervalles, il nous apparaissait intéressant de voir le rôle joué par la longueur des intervalles.



(a) Partition en 7 groupes



(b) Partition en 6 groupes

FIG. 7.11 – Classifications obtenues par *SCLUST* pour la base *iGP*.

7.3 Jeu de données *iGP bis*

Reprenons donc la base *iGP*, dont les longueurs d'intervalle ont été multipliées par 3. On remarque sur le graphique ci-dessous que les trois groupes du bas restent relativement bien isolés, alors que ceux du haut semblent peu séparables.

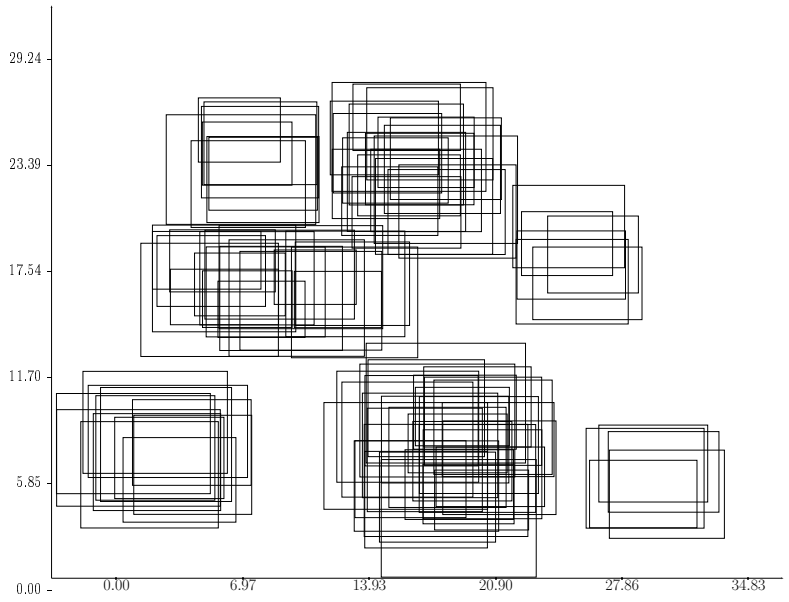
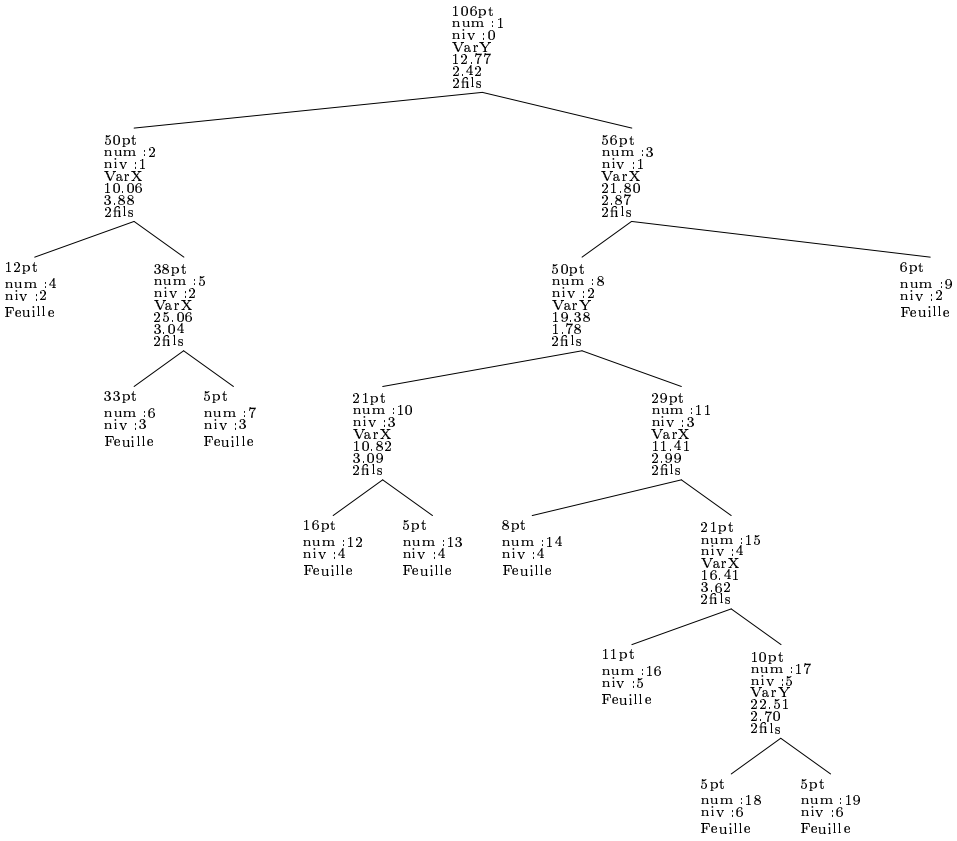


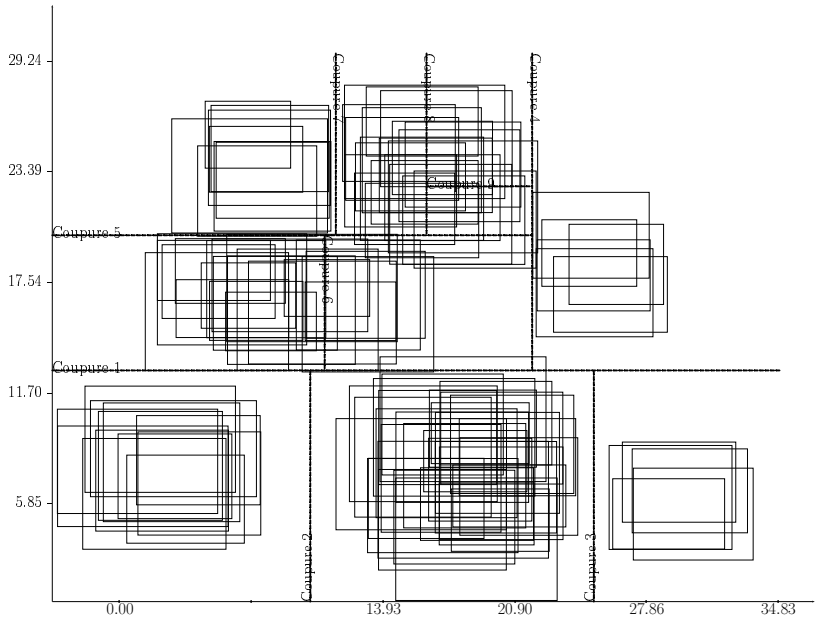
FIG. 7.12 – Présentation de la base *iGP bis*.

7.3.1 Par la méthode *SHOPP*

Comme sans utiliser le test de Fisher, la méthode effectuait plus de coupures que nécessaire, les résultats présentés ici ont été obtenus en l'utilisant. On peut remarquer que les coupures se font aux mêmes endroits qu'avec la base *iGP* initiale ; ce qui est tout à fait logique puisque les coupures se font uniquement entre les centres d'intervalles. La longueur des intervalles intervient, elle, dans la mesure de l'aire enlevée (Δ). C'est ainsi que *SHOPP*, après avoir isolé correctement les 7 classes, continue à découper la classe centrale et la classe située en haut au centre (dont l'effectif est plus conséquent que celui des autres classes à sa proximité). L'arbre de classification, ainsi qu'un graphique illustrant ces coupures, sont visibles ci-après.



(a) Arbre de classification



(b) Graphique montrant les coupures

FIG. 7.13 – Classification obtenue par *SHOPP* pour la base *iGP bis*.

Les méthodes *SCLASS* et *SCLUST* ne nous ayant pas fourni de résultats très satisfaisants sur le jeu de données initiales *iGP*, nous ne comparerons pour cette base *iGP bis* la méthode *SHOPP* qu'à la méthode *DIV*. Remarquons que la méthode *SHOPP* doit s'arrêter après avoir obtenu le bon nombre de classes, alors que *DIV* s'arrête après le nombre de classes fixé par l'utilisateur. Maintenant, on pourrait envisager d'utiliser des méthodes de détermination du nombre de classe, pour ensuite fixer le nombre de classes que *SHOPP* doit retourner.

7.3.2 Par la méthode *DIV*

En demandant 7 classes à la méthode *DIV*, celle-ci nous retourne les bonnes classes ! Remarquons qu'elle effectue précisément les mêmes coupures aux mêmes endroits qu'avec la base *iGP* initiale. Elle n'a donc pas été influencée du tout par la modification de la longueur des intervalles. Voici un graphique représentant ces coupures.

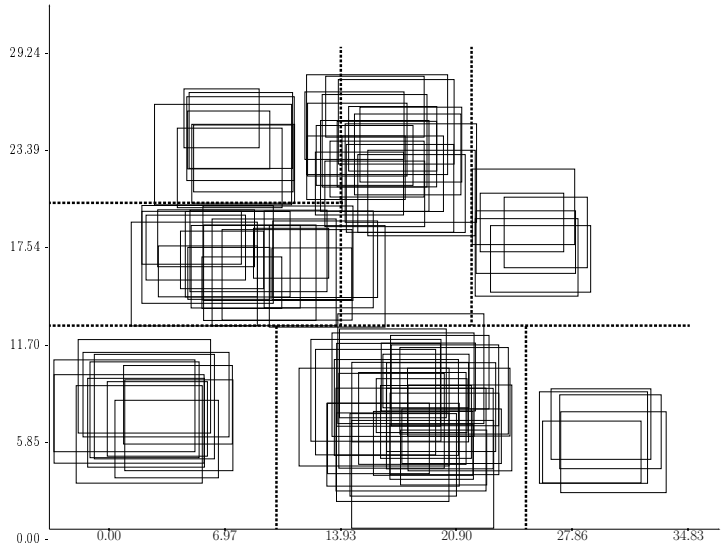


FIG. 7.14 – Classification obtenue par *DIV* pour la base *iGP bis*.

Ceci clôture notre étude de la base *iGP*. Nous allons maintenant nous pencher sur la célèbre base de *Ruspini*.

7.4 Jeu de données *iRuspini*

Nous allons donc aborder la base *Ruspini* (introduite dans [Ruspini, 1970]). Celle-ci a été transformée en données de type intervalle de la manière suivante : les variables originelles sont devenues les variables "milieu d'intervalle", et la demi-longueur de chaque intervalle a été fixé arbitrairement à 1.

Voici un graphique nous montrant ce nouveau jeu de données :

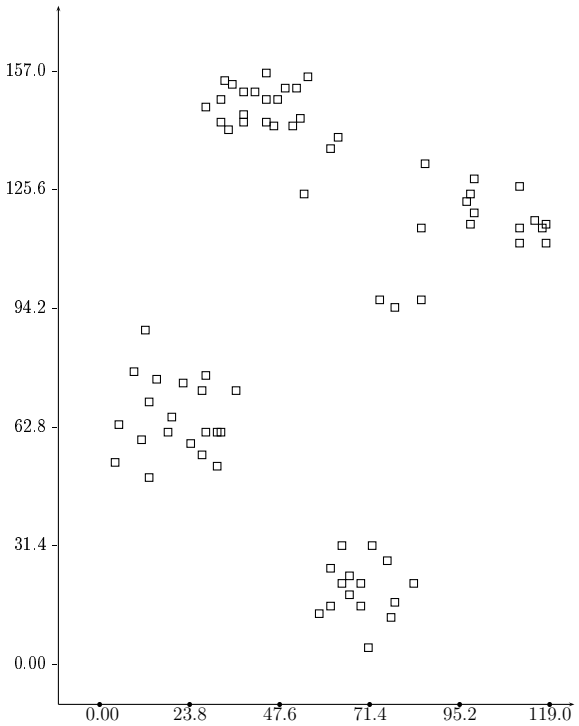


FIG. 7.15 – Présentation de la base *iRuspini*.

Cette base de 75 individus à deux variables est connue pour sa difficulté à trouver le bon nombre de classes, à cause des huit points situés entre les deux classes du haut.

7.4.1 Par la méthode *SHOPP*

Munie du test d'arrêt supplémentaire (test de Fisher), *SHOPP* retrouve 7 classes dans notre jeu de données après les coupures, et l'étape d'élagage permet d'en limiter le nombre à 5. On voit sur le graphique que la classe qui lui pose problème est celle du coin supérieur droit.

D'une part, elle est amputée des trois points situés au centre du graphique, et comme il n'y a pas d'étape de recollement, ces points restent définitivement isolés. Remarquons qu'en observant le graphique, on pourrait se demander s'il ne faut pas plutôt voir 5 classes dans ce jeu de données... D'autre part, cette classe du coin supérieur droit est ensuite coupée verticalement en trois sous-classes, mais cette branche de l'arbre de classification est finalement élaguée, et on se retrouve alors avec 5 classes.

Voici un graphique montrant les différentes coupures effectuées. Les coupures élaguées y sont tracées en pointillés, alors que les lignes pleines représentes les coupures conservées. La page suivante nous montre les arbres de classification (avant et après élagage) retournés par cette méthode.

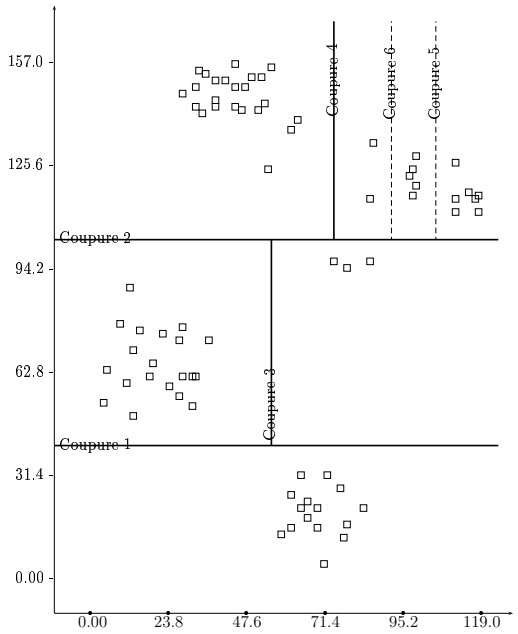
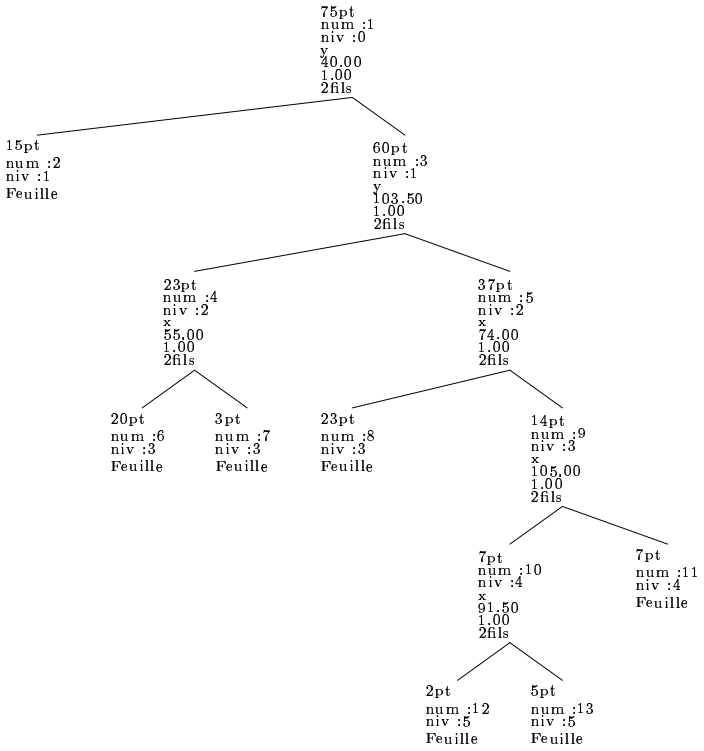
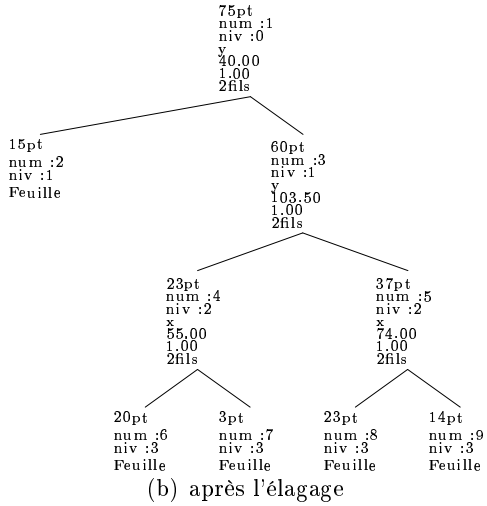


FIG. 7.16 – Résultats de la méthode *SHOPP* pour la base *iRuspini*.



(a) avant l'élagage



(b) après l'élagage

FIG. 7.17 – Arbres de classification obtenus par *SHOPP* pour la base *iRuspini*.

7.4.2 Par la méthode *SCLASS*

La méthode *SCLASS* effectue ici ses coupures aux mêmes endroits que *SHOPP*, mais elle en fait davantage. Ainsi, on a 9 classes avant l'élagage, et après on en a encore 7, qui sont précisément celles obtenues par *SHOPP* avant son étape d'élagage. Voici un graphique représentant ces différentes coupures, les traits en pointillés montrant des coupures élaguées.

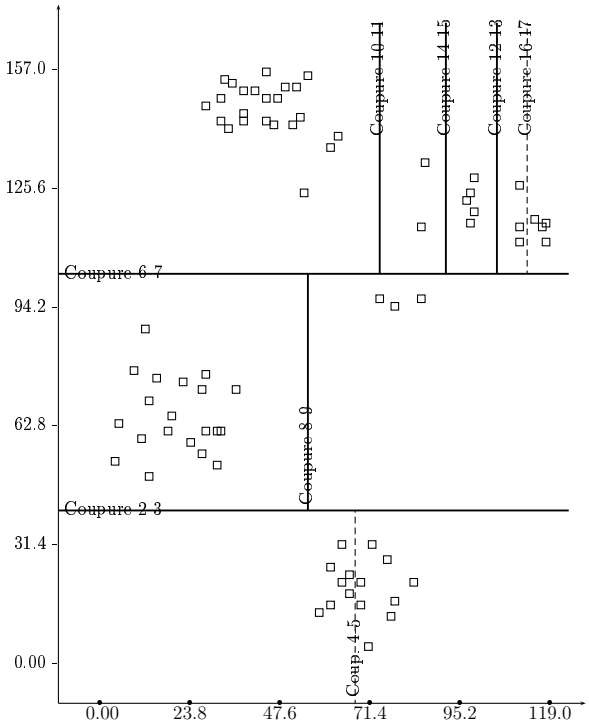


FIG. 7.18 – Résultats de la méthode *SCLASS* pour la base *iRuspini*.

Nous allons maintenant tester les méthodes *DIV*, puis *SCLUST*, qui toutes deux demandent, en argument, le nombre de classes à retourner, alors que *SHOPP* et *SCLASS* doivent déterminer elles-mêmes ce nombre en cessant leurs coupures au moment opportun.

7.4.3 Par la méthode *DIV*

Quand on lance la méthode *DIV* en lui demandant 4 classes, elle retrouve les 4 classes traditionnelles. La question intéressante est de voir ce que cette méthode propose comme classification en 5 groupes. Voici un graphique reprenant en traits continus les coupures qu'effectue *DIV* pour obtenir quatre classes, et en pointillés la coupure supplémentaire nécessaire pour en obtenir cinq.

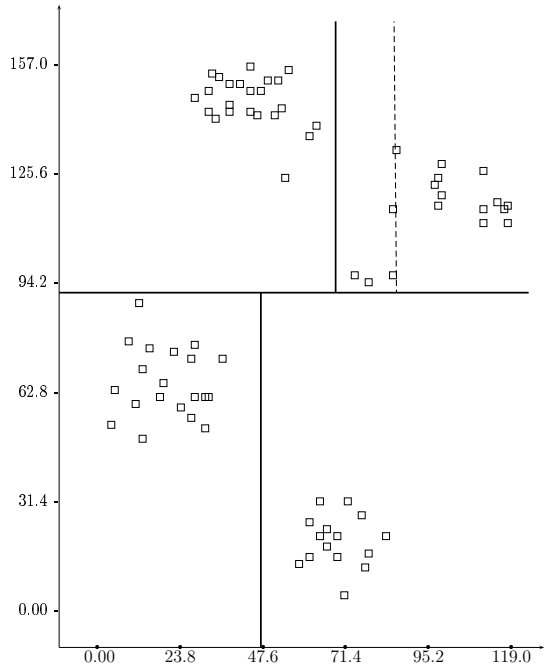


FIG. 7.19 – Résultats de la méthode *DIV* pour la base *iRuspini*.

On constate que *DIV* effectue sa première coupure de façon horizontale, mais plus bas que *SHOPP* et *SCLASS*, de manière à ne pas séparer les trois points litigieux de leur classe. Remarquons également que la partition en cinq classes n'est pas tout à fait la même qu'avec *SHOPP* et *SCLASS*. En effet, la quatrième coupure de *DIV* passe précisément entre les deux points supérieurs du "L" inversé. Cela signifie que la cinquième classe comporte ici 4 individus, et non 3 comme avec *SHOPP* et *SCLASS*.

Passons maintenant à l'étude de ce jeu de données par la méthode *SCLUST*.

7.4.4 Par la méthode *SCLUST*

La méthode *SCLUST* nous fourni ici exactement les mêmes partitions (en quatre et cinq classes) que la méthode *DIV*. Notons que cette méthode *SCLUST* travaille de façon tout à fait différente des trois autres méthodes vu que c'est une méthode de partitionnement polythétique. Cependant, elle utilise le même critère que *DIV*, à savoir l'inertie.

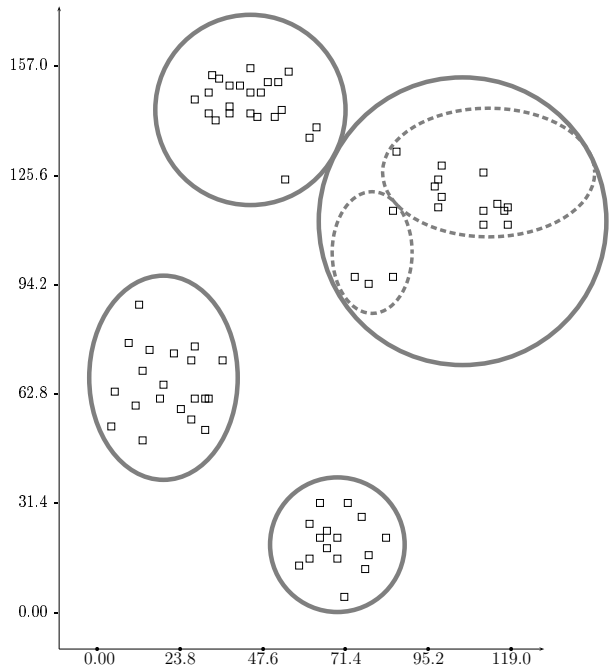


FIG. 7.20 – Résultats de la méthode *SCLUST* pour la base *iRuspini*.

Ce graphique nous montre, en traits pleins, les quatre classes proposées par *DIV*, et les traits pointillés illustrent comment cette méthode partage les points de la classe du coin supérieur droit pour obtenir une partition en cinq classes. Passons maintenant à un jeu de données dont les groupes ne peuvent pas être isolés par des coupures parallèles aux axes...

7.5 Jeu de données *iTriangle*

Nous allons maintenant étudier notre dernier jeu de données à deux variables. Comme on peut le voir sur le graphique, il se compose de trois classes allongées, qui ne sont pas directement séparables par des coupures perpendiculaires aux axes. Nos méthodes monothétiques risquent donc d'avoir du mal à retrouver les bonnes classes...

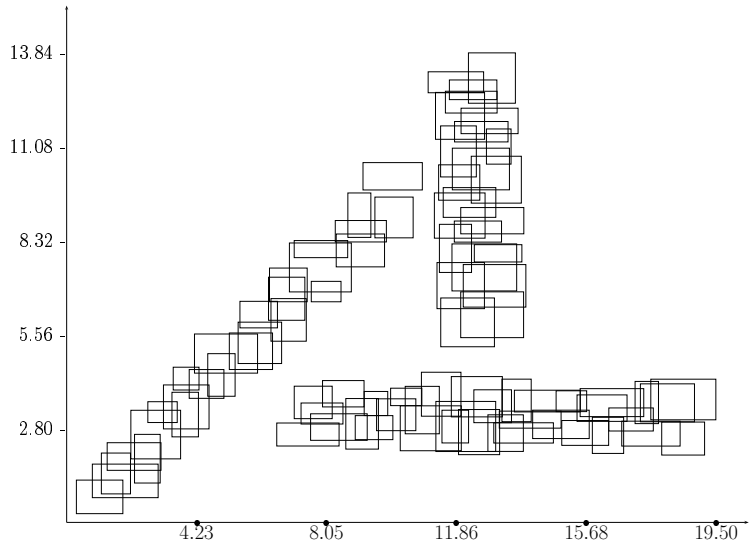
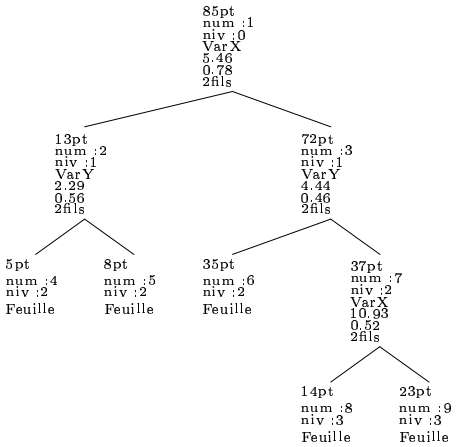


FIG. 7.21 – Présentation de la base *iTriangle*.

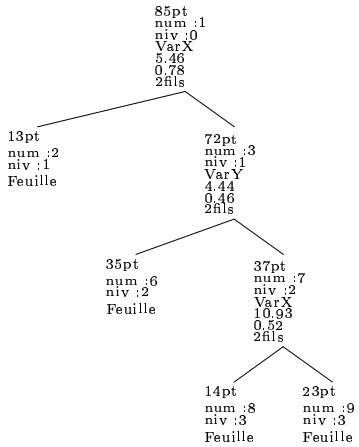
7.5.1 Par la méthode *SHOPP*

Comme on s'y attendait, *SHOPP* commence par couper dans une classe, ensuite, elle isole correctement les autres groupes. On se retrouve donc avec une partition en quatre classes. C'est dans ce genre de cas qu'il serait intéressant d'avoir une troisième étape de type "recollement", qui regrouperait les classes qui ont dû être découpées pour pourvoir en séparer d'autres. Signalons que l'étape d'élagage permet de supprimer une coupure inutile.

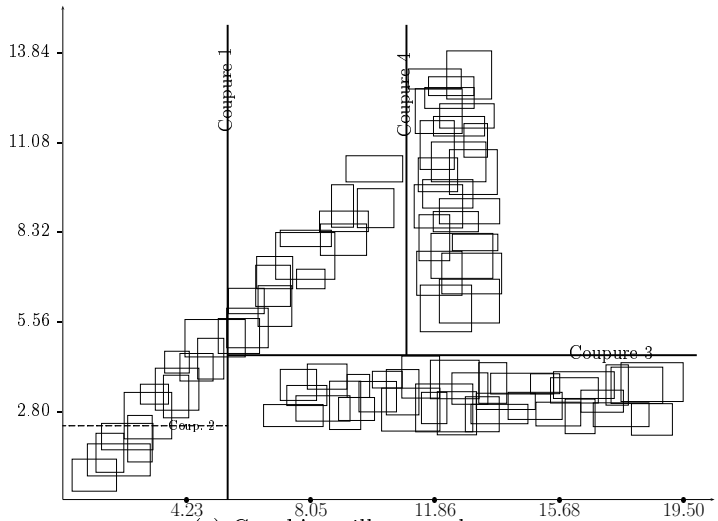
Remarquons que le test d'arrêt supplémentaire (test de Fisher) a été utilisé pour obtenir ces résultats, sans quoi on aurait obtenu bien plus de classes...



(a) Arbre de classification avant l'élagage



(b) Arbre de classification élagué



(c) Graphique illustrant les coupures

FIG. 7.22 – Classification obtenue par *SHOPP* pour la base *iTriangle*.

7.5.2 Par la méthode *SCLASS*

Le module *SCLASS* de [SODAS, 2] nous a encore posé problème pour ce jeu de données, et il a été nécessaire d'augmenter la taille minimale qu'un noeud doit avoir pour pouvoir être coupé à 12 individus, pour que *SCLASS* nous retourne des résultats, et ne plante pas !

Sur le graphique ci-dessous, on peut voir que *SCLASS* commence par couper au travers de la classe du bas, et qu'ensuite, ses autres coupures se font aussi souvent entre les groupes qu'au travers. Remarquons que si certains trous évidents dans les classes retournées par *SCLASS* (grand rectangle tout en bas, ou carré au centre) n'ont pas servi à des coupures, c'est à cause de l'effectif minimal des classes que nous avons dû élever à 12.

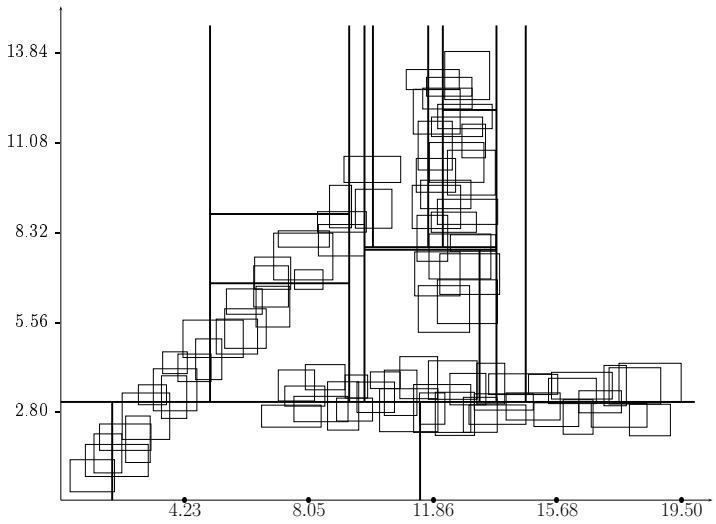


FIG. 7.23 – Résultat de la méthode *SCLASS* pour la base *iTriangle*.

Les résultats de cette méthode sont donc très décevants, mais on peut comprendre qu'une méthode monothétique ait des difficultés avec ce genre de disposition de classes. Voyons maintenant comment se débrouille *DIV*, une autre méthode monothétique.

7.5.3 Par la méthode *DIV*

La méthode *DIV* demandant le nombre de classes, nous avons essayé avec trois classes (il y a trois classes dans notre base), puis avec quatre (*SHOPP* nous proposant une partition en quatre classes). Mais cette méthode ne parvient pas non plus à retrouver les bonnes classes. Le graphique ci-dessous nous montre en traits pleins les coupures nécessaires pour obtenir trois classes, et la coupure en pointillés est celle qui nous permet d'en obtenir quatre.

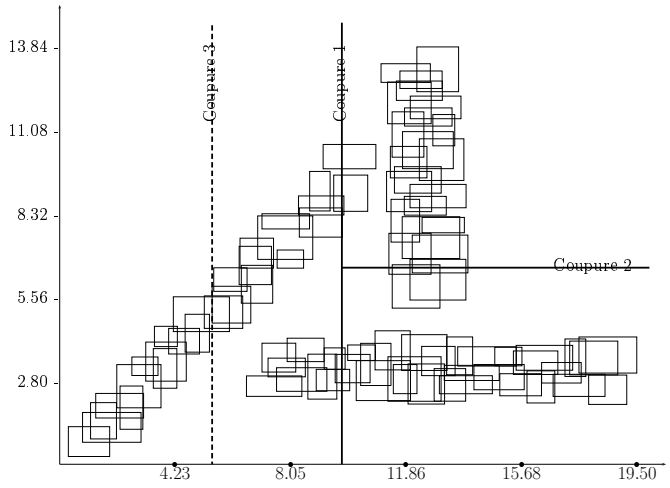


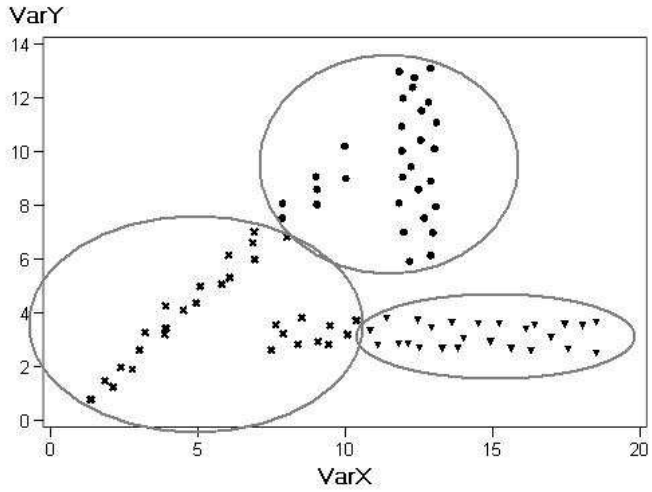
FIG. 7.24 – Résultats de la méthode *DIV* pour la base *iTriangle*.

La première coupure se fait au travers de la classe du bas, mais elle isole aussi les deux individus du haut de la classe oblique. Ensuite, la deuxième coupure se fait au travers de la classe de droite, à cause de ces deux individus. Enfin, pour obtenir quatres classes, c'est la classe oblique qui est encore coupée. Ces résultats sont donc plutôt mauvais, mais tout à fait logiques pour une méthode monothétique. Voyons maintenant ce que nous donne la méthode *SCLUST*, qui n'est pas monothétique, mais qui cependant a tendance à retourner des classes hypersphériques...

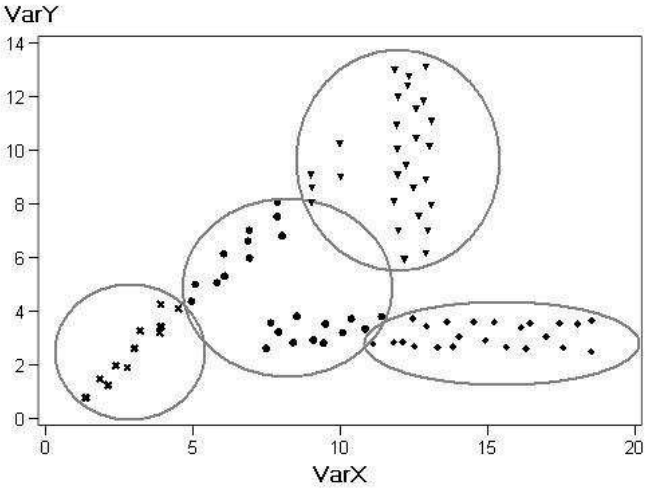
7.5.4 Par la méthode *SCLUST*

Tout comme pour *DIV*, nous avons demandé trois classes, puis quatre. Et comme prévu, *SCLUST* ne parvient par à retrouver les bonnes classes, mais essaye d'obtenir des groupes hypersphériques.

Les graphiques ci-dessous illustrent les résultats obtenus. Précisons que seuls les centres des rectangles sont représentés, pour plus de clarté, bien que les classifiactions aient été effectuées avec l'entièreté des données...



(a) Partition en 3 classes



(b) Partition en 4 classes

FIG. 7.25 – Résultats obtenus par *SCLUST* pour la base *iTriangle*.

7.6 Conclusions de l'analyse des exemples bivariés

Un fait important à noter est que les méthodes *SHOPP* et *SCLASS*, qui sont pourtant basées sur un algorithme semblable, n'effectuent généralement pas les mêmes coupures. Ainsi, il nous a paru surprenant qu'avec des jeux de données dont les classes sont relativement bien isolées, *SCLASS* coupe carrément au travers des groupes, alors que *SHOPP* parvient sans aucun problème à retrouver les bonnes classes. Il n'y a finalement qu'avec la base *iRuspini* que *SCLASS* s'en sort bien (en effectuant les mêmes coupures que *SHOPP*). L'importance des hypothèses (processus homogène ou non homogène) n'est donc pas à négliger...

Pour ce qui est de la méthode *SHOPP*, remarquons que si on laisse faire toutes les coupures (sans le test de Fisher), l'élagage n'est pas assez puissant pour parvenir à retrouver le faible nombre de classe que l'on est censé obtenir. Il apparait donc préférable d'utiliser ce test d'arrêt supplémentaire, d'autant plus que si la base est conséquente et contient peu de groupes (cas de la base *iBSP*, par exemple), la méthode va passer beaucoup de temps à effectuer de nombreuses coupures supplémentaires (pour n'avoir plus que des classes de 1 ou 2 individus), qu'il faut ensuite élaguer...
Ce test permet donc de limiter le nombre de classes finales, ainsi que le temps d'exécution.

Pour la comparaison, nous avons également utilisé la méthode de partitionnement *SCLUST*. Ses résultats ne sont pas mauvais, puisqu'elle n'a des problèmes qu'avec la base *iGP*, et la base *iTriangle* où toutes les méthodes ont eu pas mal de difficultés. Pour l'étude du jeu de données réels, nous n'utiliserons cependant plus cette méthode, pour mieux se concentrer sur les méthodes hiérarchiques divisives.

Concernant la méthode *DIV*, celle-ci s'est montrée efficace puisque hormis la base *iTriangle*, elle n'a eu de problème qu'avec un seul individu émergeant de sa classe dans la base *iBSP*. Cette méthode parvient donc, en général, aux mêmes résultats que *SHOPP*, mais en effectuant parfois des coupures différentes...

Une base qui a donc montré les limites des méthodes monothétiques, comme on vient de le dire, est *iTriangle*. Etant composée de trois classes rectangulaires disposées en triangle, il n'était pas possible de séparer les groupes directement : il fallait commencer par couper au travers d'un groupe. Et à ce petit jeu, la méthode *SHOPP* s'en est mieux sortie que les autres. Remarquons qu'il serait intéressant d'avoir une méthode de recollement qui puisse regrouper au sein d'une même classe les parties qu'il a fallu séparer.

Ceci clôture notre analyse de jeux de données artificiels à deux variables. Passons maintenant à une application plus ludique, la classification de différents modèles d'automobiles.

7.7 Jeu de données réel : *cars*

Dans cette section, nous allons étudier un jeu de données réelles concernant le monde automobile. Celui-ci se compose de trente-trois modèles vendus en 2001, sur lesquels huit variables de type intervalle ont été mesurées.

Les différents modèles que nous allons étudier sont les suivants :

0	Alfa Roméo 145	11	Fiat Punto	22	Mercedes Classe S
1	Alfa Roméo 156	12	Ford Fiesta	23	Nissan Micra
2	Alfa Roméo 166	13	Ford Focus	24	Opel Corsa
3	Aston Martin DB7	14	Honda NSK	25	Opel Vectra
4	Audi A3	15	Lamborghini Diablo	26	Porsche
5	Audi A6	16	Lancia Y	27	Renault Twingo
6	Audi A8	17	Lancia K	28	Rover 25
7	BMW Série 3	18	Maserati GT	29	Rover 75
8	BMW Série 5	19	Mercedes SL	30	Skoda Fabia
9	BMW Série 7	20	Mercedes Classe C	31	Skoda Octavia
10	Ferrari	21	Mercedes Classe E	32	Volkswagen Passat

Et les variables mesurées :

- le prix, exprimé en milliers de lires italiennes ;
- la cylindrée, exprimée en cm^3 ;
- la vitesse maximale, exprimée en km/h ;
- l'accélération, qui est le temps en secondes que met le véhicule pour effectuer un 0-100 km/h ;
- l'empattement, en cm ;
- la longueur, en cm ;
- la largeur, en cm ;
- la hauteur, en cm.

Avant toute chose, il est nécessaire de normaliser notre jeu de données. En effet, comme notre méthode recherche pour chaque variable le plus grand "trou" entre deux individus (Δ), si on ne normalise pas, *SHOPP* n'utilisera que la variable "prix" pour effectuer ses coupures, puisque les prix sont de l'ordre de la dizaine de milliers, alors que la cylindrée est de l'ordre des milliers, la vitesse et les dimensions sont de l'ordre de la centaine, et l'accélération de l'ordre de l'unité.

Ainsi, nous avons calculé l'écart-type, pour chaque variable, des 66 valeurs (33 minimum et 33 maximum), et nous avons ensuite divisé ces 66 valeurs par cet écart-type. Il nous a semblé plus judicieux de diviser la valeur minimale et la valeur maximale d'une variable par la même constante.

7.7.1 Par la méthode *SHOPP*

Nous avons pu alors lancer nos analyses, en commençant par la méthode *SHOPP* munie du test de Fisher, comme les précédents résultats se sont systématiquement révélés meilleurs avec ce critère d'arrêt supplémentaire. Et là, les résultats nous ont fort déçu : *SHOPP* n'effectue aucune coupure (et donc aucun élagage !)...

FIG. 7.26 – Résultats de *SHOPP* muni du test de Fisher, pour la base *Cars*.

Nous avons ensuite réessayé, mais avec pour seul critère d'arrêt l'effectif minimal du noeud à couper (un noeud qui ne contient plus que deux individus n'est plus coupé). On obtient alors 24 classes après les coupures, qui seront ensuite ramenées à 13 par l'étape d'élagage.

Les arbres de classification reprenant ces coupures se trouvent aux pages suivantes. Il y a l'arbre non élagué (sur deux pages horizontales), puis l'arbre élagué (vertical). On remarque que *SHOPP* a tendance à sortir de la masse les voitures les plus différentes plutôt que d'effectuer des groupes homogènes après chaque coupure. On a donc des arbres qui s'étirent tout en longueur, mais sans grande consistance horizontale...

33pt
num :1
niv :0
haut
1538.28
25.75
2fils
1pt
num :2
niv :1
Feuille
32pt
num :3
niv :1
cylind
353.71
31.91
2fils
31pt
num :4
niv :2
prix
185.99
69.85
2fils
1pt
num :5
niv :2
Feuille
28pt
num :6
niv :3
haut
1763.56
0.00
2fils
3pt
num :7
niv :3
haut
1763.56
0.00
2fils
5pt
num :8
niv :4
accel
325.43
33.46
2fils
23pt
num :9
niv :4
pat
1681.90
19.99
2fils
1pt
num :46
niv :4
Feuille
2pt
num :47
niv :4
Feuille
4pt
num :10
niv :5
cylind
266.56
57.48
2fils
1pt
num :11
niv :5
Feuille
22pt
num :16
niv :5
accel
399.19
76.79
2fils
1pt
num :17
niv :5
Feuille
3pt
num :12
niv :6
pat
1393.49
0.00
2fils
1pt
num :13
niv :6
Feuille
19pt
num :18
niv :6
long
834.97
0.00
2fils
3pt
num :19
niv :6
accel
414.39
53.98
2fils
1pt
num :14
niv :7
Feuille
2pt
num :15
niv :7
Feuille
1pt
num :20
niv :7
Feuille
18pt
num :21
niv :7
cylind
188.43
60.70
2fils
1pt
num :44
niv :7
Feuille
2pt
num :45
niv :7
Feuille
15pt
num :22
niv :8
long
1001.96
0.00
2fils
3pt
num :23
niv :8
larg
1949.15
0.00
2fils
Division
2pt
num :42
niv :9
Feuille
1pt
num :43
niv :9
Feuille

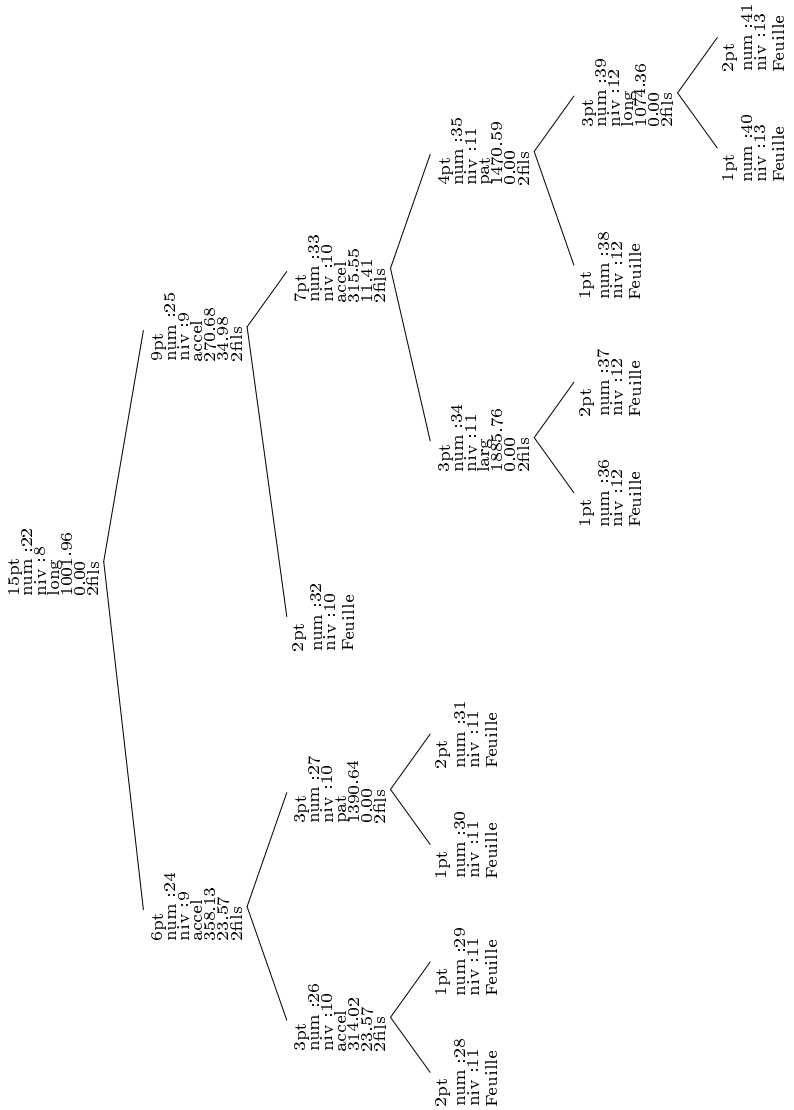


FIG. 7.27 – Résultats (avant élagage) de la méthode *SHOPP* sans test de Fisher, pour la base *Cars*.

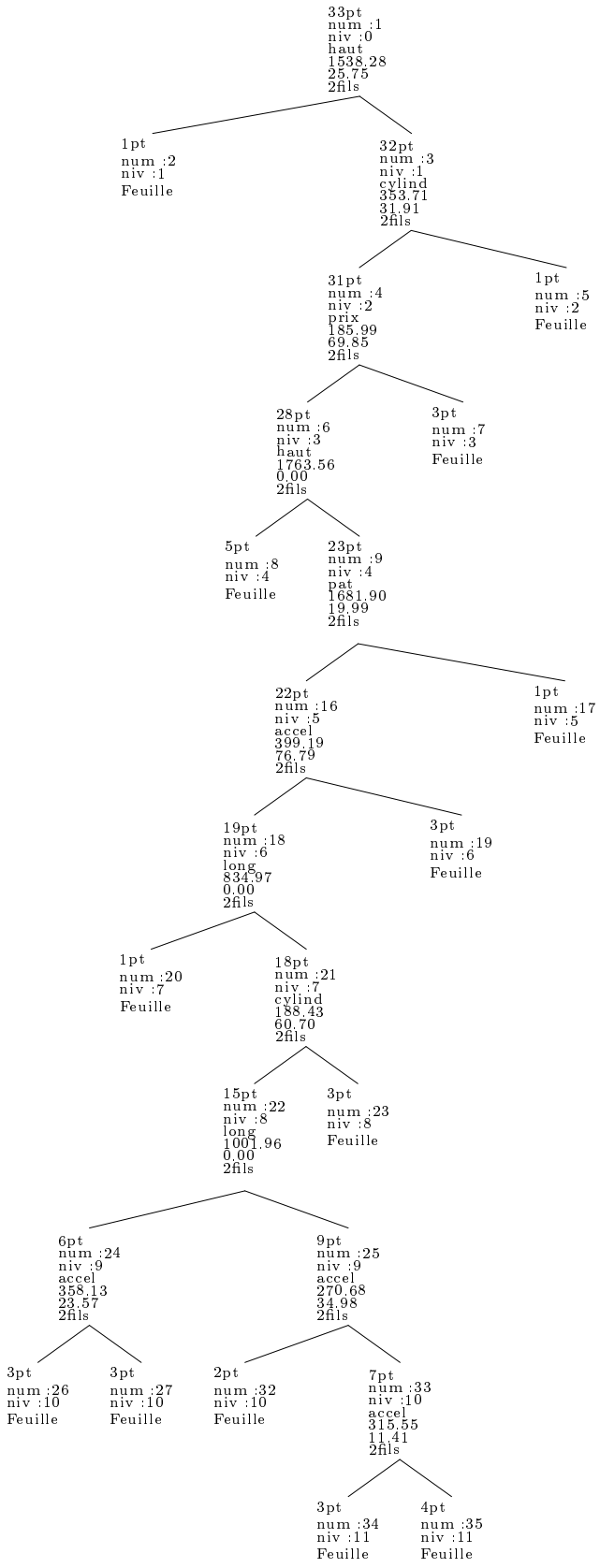


FIG. 7.28 – Résultats (après élagage) de la méthode *SHOPP* sans test de Fisher, pour la base *Cars*.

Analysons ces résultats. Tout d'abord, *SHOPP* isole la *Lamborghini Diablo* du lot, car elle est bien plus basse que les autres (111 cm, contre une moyenne de 130 cm pour les autres sportives, et 140 cm de moyenne générale). Ensuite, c'est au tour de l'*Aston Martin DB7* d'être isolée, à cause de sa cylindrée (6935 cm^3). La seule à partager ce type de motorisation est la *Lamborghini Diablo*, qui a déjà été isolée ; les autres voitures disponibles avec un moteur de six litres l'étant également avec de plus petits moteurs.

Dans ce qui reste, *SHOPP* isole ensuite les *Mercedes Classe E*, *Mercedes Classe S* et *Ferrari*, pour leur prix bien plus élevé que la moyenne... Remarquons que la *Ferrari* a été séparée des deux *Mercedes* pour sa hauteur, mais cette coupure a été élaguée.

Ensuite, c'est encore la variable *hauteur* qui intervient pour isoler les cinq plus basses voitures qui restent. Il s'agit de la *Porsche*, la *Honda NSK*, la *Maserati GT*, la *Mercedes SL* et la ... *Ford Fiesta* ! Cela peut sembler étrange, mais *SHOPP* coupe à hauteur de 137 cm, et cette *Ford* mesure 132 cm. Notons que les voitures plus hautes font au moins 142 cm. La *Fiesta* est ensuite mise à l'écart, pour son 0-100 bien plus lent que celui des quatre sportives avec lesquelles elle se trouve. Puis, c'est la *Mercedes SL* qui est isolée, pour sa cylindrée plus importante, et après, la *Porsche*, pour son empattement plus court. Néamoins, après élagage, ces cinq modèles se retrouvent dans la même classe, bien que nous aurions préféré ne pas voir figurer la *Ford Fiesta* parmi toutes ces sportives...

C'est ensuite à la *BMW Série 7* de se voir mise à part, pour son empattement allant de 293 à 307 cm ; l'*Audi A8* n'étant pas disponible en version aussi longue, et la *Mercedes Classe S* qui, elle, est un peu plus longue, est déjà mise de côté.

C'est maintenant au tour des *Fiat Punto*, *Nissan Micra*, et *Skoda Fabia* de se voir séparées du reste, pour leur accélération plutôt médiocre. Remarquons qu'avant l'élagage, la *Punto* était encore séparée des *Micra* et *Fabia*, pour avoir de moins mauvaises accélérations que ses deux consoeurs.

Puis c'est la *Renault Twingo* qui est mise à part, pour sa longueur nettement plus faible que l'ensemble des modèles étudiés ici. Signalons cependant que de part son accélération à peine meilleure d'une seconde et demi que les trois citadines isolées à l'étape précédente, il nous semble qu'elle aurait pu se retrouver avec la *Fabia*, *la Micra* et la *Punto*.

C'est ensuite la cylindrée, qui permet d'isoler les *Audi A6*, *A8* et la *BMW Série 5*. Signalons qu'avant l'étape d'élagage, l'*Audi A8* était logiquement séparée des deux autres pour sa largeur plus conséquente...

Vient ensuite le moment de séparer les voitures qui nous restent selon leur longueur. Parmi les six petites, *SHOPP* distingue celles qui ont une bonne accélération (*Audi A3*, *Alfa Roméo 145* et *Ford Focus*) des plus lentes (*Lancia Y*, *Rover 25* et *Opel Corsa*). Remarquons qu'avant l'étape d'élagage, la *Focus* était séparée de l'*Alfa* et de l'*Audi* (pour son accélération), et la *Lancia* des *Rover* et *Opel* pour son empattement plus court.

Celles qui nous restent sont donc toutes des berlines de milieu de gamme. *SHOPP* sépare alors la *BMW Série 3* et la *Mercedes Classe C* des autres, à cause de leur accélération. C'est aussi l'accélération qui permet de séparer les *Alfa 156* et *166* et la *Lancia K* des *Skoda Octavia*, *Opel Vectra*, *Rover 75* et *Volkswagen Passat*.

Signalons encore qu'avant l'élagage, l'*Alfa 156* était séparée des *Alfa 166* et *Lancia K* à cause de sa largeur, et dans le groupe des quatre dernières berlines, la *Skoda* était mise à part pour son empattement moindre, puis la *Vectra* l'était aussi, pour sa plus faible longueur.

Nous nous retrouvons donc, après élagage, avec les classes suivantes :

1		Lamborghini Diablo	
2		Aston Martin DB7	
3	Ferrari	Mercredes Classe E	Mercedes Classe S
4	Porsche Mercedes SL	Honda NSK Ford Fiesta	Maserati GT
5		BMW Série 7	
6	Fiat Punto	Nissan Micra	Skoda Fabia
7		Renault Twingo	
8	Audi A6	Audi A8	BMW Série 5
9	Audi A3	Alfa Roméo 145	Ford Focus
10	Lancia Y	Rover 25	Opel Corsa
11	BMW Série 3	Mercedes Classe C	
12	Alfa Roméo 156	Alfa Roméo 166	Lancia K
13	Skoda Octavia	Opel Vectra Volkswagen Passat	Rover 75

En observant ce tableau, ces classes nous semblent compréhensibles. Mis à part la présence de la *Ford Fiesta* parmi les sportives, on peut dire que ces 13 classes sont homogènes. Les berlines sont avec les berlines, les citadines sont regroupées dans trois classes, les berlines sportives allemandes ont leur propre classe (la 11), les berlines sportives italiennes aussi (la 12), et les compactes sportives également : elles ne sont pas casées parmi les berlines ni les citadines. La présence de la *Ferrari* parmi les deux grandes berlines *Mercedes* est quelque peu génante dans ce tableau, mais signalons qu'avant l'élagage, une séparation existait.

Maintenant, on a tout de même treize classes, ce qui est quand même beaucoup avec 33 individus. On a aussi trois classes qui ne contiennent qu'un seul élément. Il semblerait donc intéressant d'avoir une méthode de recollement qui permettrait de regrouper différentes classes. Intuitivement, en effet, on voudrait pouvoir fusionner les classes 1-2-4 pour obtenir la classe des sportives, 3-5-8 pour les berlines de luxe, 6-7-10 pour les citadines, et 11-12-13 pour les berlines de grande série...

7.7.2 Par la méthode *SCLASS*

Pour la comparaison, voici les classes (après élagage) proposées par *SCLASS* :

1	Aston Martin DB7	Honda NSK	Mercedes SL
	Ferrari	Lamborghini Diablo	Porsche
	Ford Fiesta	Maserati GT	
2	Alfa Roméo 145	Audi A8	Lancia K
	Alfa Roméo 156	BMW Série 3	Mercedes Classe C
	Audi A3	BMW Série 5	Mecredes Classe E
	Audi A6	BMW Série 7	Mercedes Classe S
3		Opel Corsa	
4	Alfa Roméo 166	Rover 25	Volkswagen Passat
	Opel Vectra	Rover 75	
5	Fiat Punto	Nissan Micra	Skoda Fabia
	Lancia Y	Renault Twingo	
6	Skoda Octavia	Ford Focus	

On remarque de suite de *SCLASS* élague beaucoup plus que *SHOPP*, puisqu'on n'a ici que six classes. Cette méthode commence par mettre de côté les 8 voitures les plus basses, et donc on se retrouve avec la *Ford Fiesta* parmi les sportives ! Notons que ce choix de variable est plus judicieux que celui de *SHOPP* puisqu'il permet de regrouper toutes les sportives au sein d'une même classe.

La coupure suivante sépare les 12 voitures les plus rapides de 0 à 100 km/h des autres. Cette classe regroupe toutes les berlines plutôt sportives et plutôt luxueuses que sont les *Alfa Roméo*, *Audi*, *BMW*, *Lancia* et *Mercedes*. Cette classe, qui est la plus grande, nous semble relativement hétérogène, puiqu'on y retrouve des compactes sportives comme l'*Audi A3* ou l'*Alfa 145* et des limousines bien plus luxueuses et plus chères comme l'*Audi A8*, la *BMW Série 7* ou les *Mercedes Classe E* et *S*.

Après avoir isolé l'*Opel Corsa*, *SCLASS* nous retourne une classe contenant 5 autres voitures ayant une accélération relativement bonne. Il s'agit des *Alfa 166*, *Opel Vectra*, *Rover 25* et *75* et de la *Passat*. La présence de la *Rover 25* dans ce groupe de berline de grande série semble quelque peu inadaptée...

Puis, par la variable "prix", *SCLASS* crée un groupe contenant toutes les citadines (sauf la *Fiesta* et la *Rover 25* donc) et un autre contenant juste la *Skoda Octavia* et la *Ford Focus*.

Voici l'arbre de classification (élagué) obtenu par la méthode *SCLASS* :

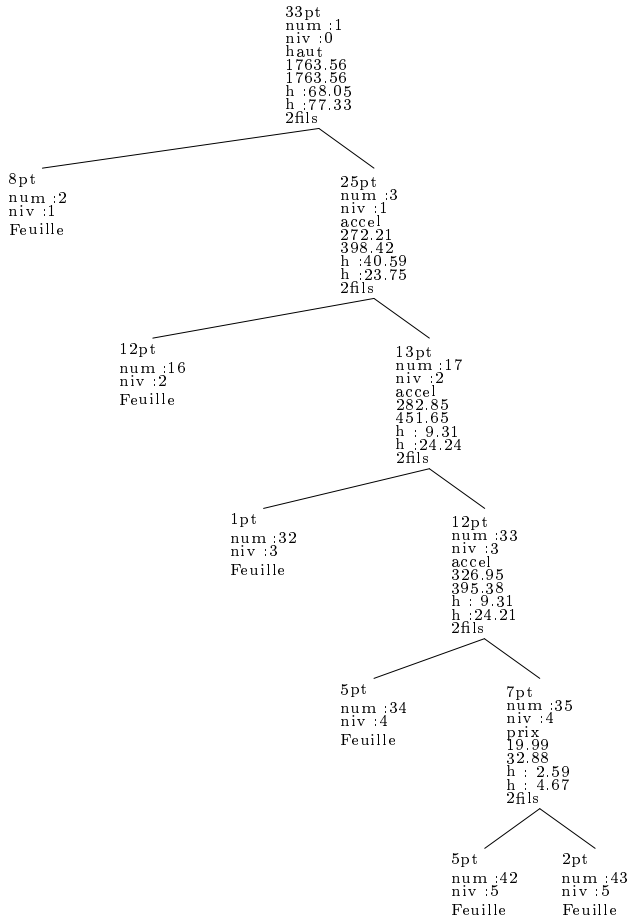


FIG. 7.29 – Résultats (après élagage) de la méthode *SCLASS*, pour la base *Cars*.

Les classes sont donc moins nombreuses, les sportives sont toutes regroupées dans une même classe, mais petites et grandes voitures sont relativement mélangées... Voyons pour terminer ce que nous propose la méthode *DIV*.

7.7.3 Par la méthode *DIV*

La méthode *DIV* commence par couper selon la variable "prix", puis divise les plus chères selon leur hauteur, et les meilleures marché selon leur longueur. De cette manière, on obtient quatre classes qui sont précisément les citadines, les berlines, les sportives et les limousines. La *Ford Fiesta* n'est donc plus considérée comme une sportive ! Cette méthode parvient ici à effectuer des divisions au sein des groupes, alors que les précédentes avaient plutôt tendance à écarter de l'analyse les modèles les plus différents...

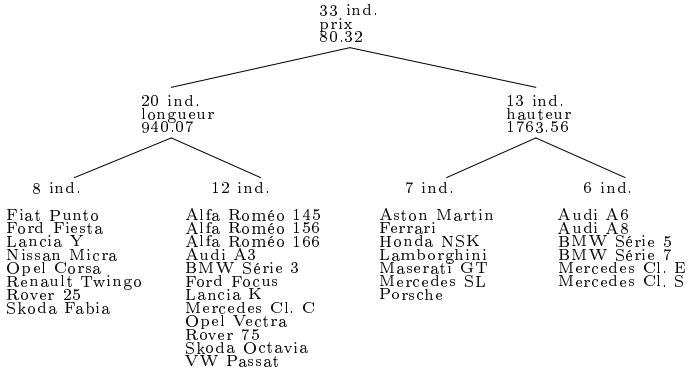


FIG. 7.30 – Résultats de la méthode *DIV* limitée à 4 classes, pour la base *Cars*.

Si on poursuit les coupures jusqu'à obtenir 13 classes, on se rend compte que *DIV* retrouve des divisions que l'on avait obtenues avec *SHOPP* ou *SCLASS*. Par exemple, les *Audi A6, A8* et *BMW Série 5* forment une classe à part, la *BMW Série 3* et la *Mercedes Classe C* aussi. La *Lamborghini Diablo* et l'*Aston Martin* forment à elles seules chacune une classe, etc.

7.7.4 Conclusions

Si on veut obtenir 4 classes, il vaudra mieux prendre la méthode *DIV*, puisque pour ce jeu de données, c'est elle qui choisi les meilleures variables de coupure. En effet, elle parvient à séparer les voitures en 2 groupes, qui sont ensuite chacun coupés en deux. Les méthodes *SCLASS* et surtout *SHOPP* choisissent, elles, plutôt des variables qui permettent d'obtenir après chaque coupure une feuille (parfois même avec un seul individu) d'un côté, et la suite de l'arbre de l'autre.

La méthode *DIV* pourrait, après l'étude de cette base, sembler meilleure, mais n'oublions pas qu'en statistique, il n'existe pas de méthode unique qui soit meilleure que toutes les autres. En considérant nos premières bases (bivariées), on s'aperçoit que pour certaines bases où *SHOPP* n'a aucun problème, *DIV* ne parvient pas à retrouver les bonnes classes, que ce soit d'un point (base *iBSP*), ou complètement (base *iTriangle*).

Signalons aussi que la méthode *SCLASS*, qui semblait couper n'importe comment dans la plupart de nos jeux de données bivariés, parvient ici à une classification relativement correcte... Un point noir la concernant est son implémentation au sein du logiciel [SODAS, 2]. En effet, au cours de nos nombreux tests, nous avons rencontré plusieurs plantages systématiques et autres bugs, dont ne sont pas affectées les autres méthodes (*DIV* et *SCLUST*). Un débuggage nous paraît nécessaire...

Un atout de la méthode *SHOPP* qu'il me semble important de souligner est sa rapidité d'exécution, surtout comparée à la méthode *SCLASS*, qui fonctionne de la même manière, mais qui doit en plus réestimer les intensités de processus avant chaque coupure.

Remarquons aussi, après avoir comparé ces différentes méthodes monothétiques, que travailler variable par variable est quelque chose de très facile, qui permet une interprétation simple, mais qui peut nous causer quelques surprises : la *Ford Fiesta* est bien une citadine, toutes ses variables prennent des valeurs semblables à celles des autres citadines, excepté une, et c'est justement cette variable qui est choisie par *SHOPP* et *SCLASS* pour effectuer une de leurs premières coupures. Notons que si la méthode *DIV* ne tombe pas dans ce panneau, c'est qu'elle utilise pour ses coupures monothétiques un critère qui, lui, fait intervenir toutes les variables : l'inertie.

Chapitre 8

Conclusions

Le sujet de ce mémoire était la généralisation aux données intervalles d'une des cinq méthodes de classification présentées par [Pirçon, 2004]. Nous avons donc, dans un premier temps, présenté les bases de la classification, puis nous avons introduit différentes notions théoriques qui intervenaient dans ces cinq méthodes. Après avoir détaillé le fonctionnement de ces méthodes, nous avons introduit le contexte des données symboliques, et plus particulièrement celles de type intervalle. Nous avons alors présenté quelques méthodes de classification symbolique, puis nous avons détaillé le fonctionnement de notre nouvelle méthode, appelée *SHOPP*, pour *Symbolic Homogeneous Poisson Process*.

Celle-ci est une méthode hiérarchique divisive monothétique, qui suppose les données issues de processus de Poisson homogènes. Elle se compose de deux étapes. La première, dite de coupure, crée un arbre de classification en effectuant différentes coupures parallèles aux axes. Ces coupures cessent quand les noeuds ne contiennent plus assez de points, mais il est possible d'utiliser un test d'arrêt supplémentaire (test de Fisher). La seconde étape, dite d'élagage, effectue un test statistique (le Gap Test) pour chaque coupure qui permet de savoir si celle-ci est judicieuse ou non. Une fois que toutes les coupures sont indexées "bonnes" ou "mauvaises", la méthode supprime tous les bouts de branche (de l'arbre de classification) qui ne contiennent que des mauvaises coupures.

Par le passé, une autre des cinq méthodes de [Pirçon, 2004] a déjà été adaptée pour traiter des données intervalles. Celle-ci est d'ailleurs disponible dans le logiciel [SODAS, 2], sous le nom de *SCLASS*. Notre adaptation s'est donc faite de manière semblable, en modélisant les intervalles par des coordonnées "milieu - demi-longueur", et en n'effectuant les coupures que selon les milieux d'intervalles.

Nous avons alors développé une application en *C++*, de près de 4000 lignes de codes (commentaires inclus), nous permettant de tester notre nouvelle méthode et de la comparer à d'autres méthodes de classification symbolique disponibles dans

le logiciel [SODAS, 2]. Pour pouvoir convertir nos simples fichiers textes au format sodas, une autre application en *C++* a été mise au point. L'utilisation de ces deux programmes est détaillée en annexe.

Nos différents tests nous ont permis de montrer que cette nouvelle méthode était tout à fait fonctionnelle et permettait d'obtenir de très bons résultats. Nos jeux de données à deux variables nous ont permis de bien visualiser comment elle se comportait, puis une application réelle, issue du monde automobile, nous a permis de montrer qu'elle parvenait à gérer des jeux de données plus complexes.

Nous tenons à mettre en évidence la simplicité de cette méthode, qui lui permet d'obtenir ses résultats relativement rapidement, surtout comparée à sa cousine *SCLASS* qui doit évaluer de nombreuses fois l'intensité de processus de Poisson. Signalons aussi la facilité d'interprétation de nos résultats, ce que l'on doit au caractère monothétique de notre méthode. Il est donc très aisé de caractériser les différentes classes obtenues, puisqu'elles sont définies par une conjonction de critères logiques.

Signalons que notre méthode *SHOPP* peut également gérer des données classiques. Il suffit en effet de présenter des jeux de données intervalles dont la longueur est nulle ! De part son critère, on remarque alors que la méthode *SHOPP* se comporte alors comme la méthode *HOPP* dont elle est une généralisation.

Pour ce qui est des perspectives d'évolution de notre méthode, il nous semblerait intéressant de lui ajouter une étape de type recollement, comme celle-ci existe dans les cinq méthodes classiques de [Pirçon, 2004]. L'absence de méthode de ce genre s'est d'ailleurs ressentie dans l'un ou l'autre de nos exemples. Cependant, son fonctionnement actuel ne nous semble pas très efficace d'un point de vue algorithmique. C'est d'ailleurs pourquoi il a été décidé de ne pas l'inclure dans *SHOPP*. Elle ne se trouve de toute façon pas dans *SCLASS*. Une autre manière d'effectuer ce recollement nous apparait donc intéressante à rechercher. On pourrait par exemple relancer une classification non plus sur les objets à étudier, mais sur les classes obtenues après élagage. On pourrait également reprendre le recollement de [Pirçon, 2004], mais en remplaçant son Gap Test par un test de Fisher, qui ne s'embarrasse pas, lui, d'enveloppes convexes ...

Un autre fait que nos tests ont mis en évidence est la limite des méthodes monothétiques. Il semblerait intéressant de pouvoir envisager des rotations, bien que celles-ci nuiraient à la facilité d'interprétation...

En conclusion, nous avons donc adapté aux données symboliques une méthode proche de *SCLASS*, mais qui fait, elle, l'hypothèse que les données suivent un processus de Poisson homogène. Cette perte d'information (la densité du processus) ne l'empêche pas d'obtenir des résultats aussi bons, voire meilleurs que ceux de *SCLASS*, et ce bien plus rapidement.

Bibliographie

[Akaike, 1954] Akaike, H. (1954). *An approximation to the density function.* Annals of the Institute of Statistical Mathematics, 6 :127-132.

[Bouget and Viénot, 1983] Bouget, D. and Viénot, A. (1983). *Traitement de l'information : statistiques et probabilités.* Librairie Vuibert.

[Chavent, 1997] Chavent, M. (1997). *Analyse des Données Symboliques : une méthode divisive de Classification.* PhD thesis, Université de Paris (X - Dauphine).

[Cox and Isham, 1980] Cox, D. R. and Isham, V. (1980). *Point Processes.* Chapman and Hall, London.

[Daudin et al., 1988] Daudin, J.-J., Masson, J.-P., Tomassone, R., and Danzart, M. (1988). *Discrimination et classement.* Masson, Paris.

[Delecroix, 1983] Delecroix, M. (1983). *Histogrammes et estimation de la densité.* Que sais-je ?

[Hardy, 2005] Hardy, A. (2005). *Cours de classification.* Facultés Universitaires N-D de la Paix, Namur.

[Hardy and Rasson, 1982] Hardy, A. and Rasson (1982). *Une nouvelle approche des problèmes de classification automatique.* Statistiques et Analyse des Données, 7, 41-56.

[Hartigan, 1975] Hartigan, J. A. (1975). *Clustering Algorithms.* John Wiley and Sons.

[Jain and Dubes, 1988] Jain, A. and Dubes, R. (1988). *Algorithms for Clustering Data.* Prentice-Hall.

[Jobson, 1991] Jobson, J. (1991). *Applied Multivariate Analysis. Volume I : Regression and Experimental Design.* Springer-Verlag.

[Jobson, 1992] Jobson, J. (1992). *Applied Multivariate Analysis. Volume II : Categorical and Multivariate Methods.* Springer-Verlag.

[Kaufman and Rousseeuw, 1990] Kaufman, L. and Rousseeuw, P. (1990). *Finding Groups in Data - An Introduction to Cluster Analysis.* Wiley, New York.

[Kubushishi, 1996] Kubushishi, T. (1996). *On some Applications of the Point Process Theory in Cluster Analysis and Pattern Recognition.* PhD thesis, Facultés Universitaires N-D de la Paix, Namur.

[Lechevallier, 2006] Lechevallier, Y. (2006). *Cours de Questions Spéciales.* Facultés Universitaires N-D de la Paix, Namur.

[Moore, 1984] Moore, D. (1984). *On the estimation of a convex set.* The Annals of Statistics, 12,3, 1090-1099.

[Parzen, 1962] Parzen, E. (1962). *On the estimation of a probability density function and the mode.* Annals of the Institute of Statistical Mathematics, 33 :1065-1076.

[Pirçon, 2004] Pirçon, J.-Y. (2004). *La classification et les processus de Poisson pour de nouvelles méthodes monothétiques de partitionnement.* PhD thesis, Facultés Universitaires N-D de la Paix, Namur.

[Pirçon and Rasson, 2002] Pirçon, J.-Y. and Rasson, J.-P. (2002). *Les arbres de Clustering.* Publications du Département de Mathématiques, n^o 2002/03, Facultés Universitaires N-D de la Paix, Namur.

[Rasson, 1976] Rasson, J.-P. (1976). *De quelques problèmes d'entropie et d'inférence pour des processus ponctuels.* PhD thesis, Facultés Universitaires N-D de la Paix, Namur.

[Rasson, 1978] Rasson, J.-P. (1978). *Estimation de formes convexes du plan.* Statistiques et Analyse des Données, 1 : 31 - 46.

[Rasson and Granville, 1996] Rasson, J.-P. and Granville, V. (1996). *Geometrical tools in classification.* Computational Statistics and Data Analysis, 23 : 105 - 123.

[Rasson et al., 2005] Rasson, J.-P., Pirçon, J.-Y., Lallemand, P., and Adans, S. (2005). *Unsupervised Divisive Classification.* Facultés Universitaires N-D de la Paix, Namur.

[Ripley and Rasson, 1977] Ripley, B. D. and Rasson, J.-P. (1977). *Finding the Edge of a Poisson Forest.* Journal of Applied Probability, 14 : 483 - 491.

[Rosenblatt, 1956] Rosenblatt, M. (1956). *Remarks on some nonparametric estimates of a density function.* Annals of the Institute of Statistical Mathematics, 27 :832-837.

[Ruspini, 1970] Ruspini, E. M. (1970). *Numerical methods for fuzzy clustering.* Information Sciences, 2 :319-350.

[Schmitt and Mattioli, 1993] Schmitt, M. and Mattioli, J. (1993). *Morphologie Mathématique.* Masson, Paris.

[Silverman, 1981] Silverman, B. W. (1981). *Using kernel density estimates to investigate multimodality.* Journal of the Royal Statistical Society, B, 70 :232-245.

[Silverman, 1986] Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis.* Chapman and Hall, London.

[SODAS, 2] SODAS (2). http ://www.info.fundp.ac.be/asso/.

Annexe A

Le programme *SHOPP*

Cette annexe présente la structure du programme de la méthode *SHOPP*, et en détaille l'utilisation.

A.1 Structure du programme

Le langage de programmation utilisé est le *C++*, compilé sous *Windows XP* avec *Dev C++*. Il peut cependant être compilé sur un système *Linux*. Le programme se compose de près de 4000 lignes de code (commentaires compris).

Voici, dans l'ordre alphabétique, les différents fichiers sources du programme :

- **clustering_tree.cxx :** il s'agit de la partie principale du programme. C'est elle qui fait appel à toutes les autres routines ;
- **coupure_multi.cxx :** contient la routine principale de coupure ;
- **coupure_multi_routine.cxx :** contient différentes sous-routines nécessaires aux coupures ;
- **declaration.h :** contient la définition des différentes constantes que l'on utilise, avec leur valeur, ainsi que nos structures de paramètres ;
- **elagage.cxx :** contient les différentes routines nécessaires à l'élagage ;
- **general.cxx :** contient plusieurs routines générales, notamment des routines de tri, les routines d'affichage, ainsi que les différentes routines de gestion des sorties LaTeX ;
- **initialisation.cxx :** contient la routine d'initialisation de l'arbre ;
- **lecture.cxx :** contient les deux routines de lecture. La première lit le fichier de données, et interroge l'utilisateur sur les différents paramètres à utiliser, la seconde se charge de stocker en mémoire une table de Fisher.

Signalons que l'exécutable du programme doit être accompagné, du fichier texte **table_F_completee.txt**. Le programme en a besoin pour effectuer le test de Fisher, et plante s'il ne le trouve pas.

Détaillons maintenant l'utilisation du programme ...

A.2 *SHOPP*, mode d'emploi

Le programme se lance d'un simple double-clic sur son icone :

Il a besoin d'avoir dans son dossier le fichier texte **table_F_completee.txt** pour tourner. Une sortie **PostScript** est effectuée, il est donc nécessaire d'avoir préalablement installé un compilateur LaTeXpour pouvoir transformer le fichier **tex** en un fichier **dvi**, puis **ps**. L'interface homme-machine se fait uniquement via la console de commande, dont voici l'affichage :

```
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --


 #########          ###
###               ##########   ##########    #########   ##########
########          ###    #### ####    ###   ###    #### ####    ###
      ###   ####   ###    ###  ####    ###  ####    ###  ####    ###
########          ###    ###  ##########   ###########  ##########
                                           ###          ###
                                          ###          ###
####################  SYMBOLIC # HOMOGENEOUS # POISSON # PROCESS ##


 Nom du fichier de donnees ? (sans l'extension '.txt')
 >
```

Le fichier de données demandé ici doit être au format texte standard. Un test est effectué pour vérifier l'existence du fichier, et il est possible d'entrer un autre nom si on s'est trompé (il faut alors préciser l'extension).

```
Comment sont présentés les intervalles ?
 0 : milieu - demi longueur
 1 : milieu - longueur
 2 : borne inf - borne sup
>
```

Le fichier doit contenir deux colonnes par variable à étudier. Celles-ci peuvent être le milieu et la demi-longueur des intervalles, le milieu et la longueur, ou les bornes inférieures puis supérieures. Les données sont alors transformées en milieu - demi-longueur pour la suite de l'analyse.

```
Y a-t-il identification des individus ? (1=OUI,0=NON)
>
```

Le fichier de données peut contenir une dernière colonne (d'entiers) permettant d'identifier les observations ou les groupes a priori. Cet identifiant est alors ajouté aux différentes sorties pour en faciliter l'interprétation.

```
Y a-t-il, en premiere ligne, le nom des variables ?
(Un seul  nom pour les deux colonnes) (1=OUI,0=NON)
>
```

Le fichier de données peut contenir, sur sa première ligne, le nom des variables. Ces noms se retrouvent alors également dans les différentes sorties. S'ils ne sont pas précisés, *SHOPP* les nommera var :0, var :1, etc.

```
Nombre de variables, ou une variable est constituee d'une colonne Centre
suivie d'une colonne Demi-Longueur ?    (s'il y a identification des
individus, ne pas inclure cette variable dans le nombre des variables!)
>
```

Le programme demande le nombre de variables intervalles (chaque variable est représentée par deux colonnes) présentes dans le fichier. La colonne d'identification ne doit pas être comptée.

Si le fichier ne contient que deux variables, le programme se propose d'ajouter aux sorties un graphique fait de rectangles, qui présente le jeu de données.

```
Voulez-vous un graphique des données ? (1=OUI,0=NON)
>
```

Le programme demande ensuite le nom que vont porter les sorties.

```
Nom des fichiers de resultats ? (sans extension)
>
```

L'extension ne doit pas être précisée, car il y a plusieurs fichiers résultats (notamment une sortie LaTeX) ayant des extensions différentes. Si les fichiers de résultats existent déjà, le programme demande s'il faut les compléter, les écraser, ou utiliser un autre nom.

```
Commentaire ? (sans espaces)
>
```

Un commentaire sous forme de chaîne de caractères sans espaces peut être inclu aux fichiers de résultats.

```
Nombre d'individus : xxx

Voulez une sortie par classe? (1=OUI, 0=NON)
>
```

On reçoit le nombre d'individus présents dans le fichier de données, puis le programme demande s'il doit effectuer une sortie par classe. Celle-ci reprend les données initiales, et y ajoute une nouvelle colonne tout à droite qui précise dans quelle classe *SHOPP* a classé les individus.

```
Quel sera le critere d'arret de coupure ?
 1=test F,
 2=Pas de test d'arret autres que le nombre de point
>
```

Le programme demande alors s'il doit utiliser le test de Fisher comme test d'arrêt des coupures, ou s'il ne doit s'arrêter qu'une fois que toutes les classes ne contiennent plus qu'un ou deux individus.

C'est alors que l'analyse commence réellement. Une fois les étapes de coupure et d'élagage terminées, le programme compile sa sortie LaTeX et se ferme. Détaillons maintenant le contenu des fichiers de résultat.

A.3 Analyse des résultats

Prenons le cas où nous aurions appelés nos fichiers de sortie "**exit**". Le premier fichier créé est le fichier **exit.txt**. Celui-ci contient le nom des autres fichiers de résultats, ainsi que celui du fichier de données. Il précise alors les différents paramètres utilisés, et donne quelques informations sur les données. Viennent ensuite le *log* de la création de l'arbre, un affichage basique de cet arbre, le *log* de son élagage, puis l'arbre élagué est réaffiché.

Un fichier **exit.arb.txt** est également créé et reprend l'arbre de classification avant et après son élagage, sous un format là aussi très simple et moyennement clair.

Viennent ensuite les fichiers **exit.tex**, **exit.dvi** et **exit.ps**. Ceux-ci contiennent un affichage intuitif et bien lisible de l'arbre de classification (avant et après élagage). Pour une meilleure lisibilité, seulement 5 niveaux de l'arbre sont représentés sur une page. Les arbres plus grands sont tronqués, et affichés sur plusieurs pages.

Enfin, si on a demandé une sortie par classe, un fichier **exit.class.txt** est créé. Celui-ci reprend en colonnes les milieux et les demi-longueurs des différentes variables, puis la colonne d'identifiant, si elle existait dans le fichier de données, et ajoute une nouvelle colonne spécifiant la classe. Remarquons que les lignes sont triées par ordre croissant selon la classe d'appartenance.

Annexe B

La Manipulation des données

Cette annexe détaille comment nous avons obtenu les données, et les transformations qu'il a fallu leur faire subir pour pouvoir être acceptées par notre application *SHOPP*, et par *SODAS*.

B.1 Obtention des jeux de données

Pour ce qui est des jeux de données bivariés, il s'agit de simples bases test (données traditionnelles) utilisées aux cours de séances de travaux dirigés et aux travaux pratiques du cours de Classification ([Hardy, 2005]). Il a donc été nécessaire de les rendre symboliques ... Pour ce faire, des valeurs ont été tirées au sort par une application *JAVA*, et *Excel* a été utilisé pour obtenir les jeux de données sous la forme nécessaire, à savoir le nom des variables sur la première ligne, puis 4 ou 5 colonnes de valeurs. La première de ces colonnes étant la première colonne d'origine, la seconde comprenant des valeurs aléatoires représentant la demi-longueur des intervalles, la troisième étant la seconde colonne d'origine, et la quatrième contenant d'autres valeurs aléatoires. Une cinquième colonne a parfois été ajoutée, contenant une indication sur la classe d'appartenance.

Les colonnes de notre fichier doivent être séparées par un (ou plusieurs) espace(s). Nos fichiers excel ont donc été sauvés au format ".prn", format texte dont le caractère séparateur est l'espace. L'extension des fichiers a ensuite été remplacée par ".txt".

Remarquons une subtilité de ce passage "texte ↔ excel". Le symbole des décimales est la virgule pour *Excel*, alors que pour le C++ (donc pour le programme *SHOPP*), pour *SODAS*, pour *SAS* (utilisé pour l'un ou l'autre graphique), il faut utiliser un point. Il a donc été nécessaire, à de nombreuses reprises, de remplacer dans le bloc-note ou le wordpad les points par des virgules puis, après avoir transformé les données avec *Excel*, les virgules par des points ...

Concernant le jeu de données réelles *cars*, celui-ci est joint au logiciel [SODAS, 2]. Cependant, ce fichier n'est disponible qu'aux formats sodas ".xml" et ".sds". Il a donc fallu en extraire les données. Pour ce faire, nous avons ouvert le fichier *cars.sds* avec un éditeur de texte.

Après de multiples entêtes, présentation des individus puis des variables, se trouve la matrice des données. Nous avons alors remis sur une seule ligne les différentes informations qui concernaient un même individu (suppression de ces "retour à la ligne"). Puis nous avons collé cette matrice dans une feuille excel, et nous avons obtenu les colonnes en utilisant la fonction "Données/Convertir", en déclarant que les colonnes étaient déterminées par le caractère " ;" ou " :". Il a ensuite fallu supprimer toutes les parenthèses (option "remplacer" dans "Edition/Rechercher", et remplacer les parenthèses par un caractère vide). Remarquons que ce jeu de données présentait trois variables qui n'étaient pas de type intervalle, celles-ci ont été supprimées.

Le jeu de données à ensuite été normalisé. Nous avons utilisé *Excel* pour calculer les écarts-types, puis diviser les bornes des intervalles par leur écart-type. Enfin, le jeu de données a, lui aussi, été sauvegardé au format ".prn", puis renommé en ".txt".

Nous avons donc obtenu différents jeux de données symboliques pour tester notre programme. Cependant, pour en comparer les résultats, il fallait pouvoir importer nos bases dans [SODAS, 2]. Pour ce faire, nous avons développé une autre application en C++, **txt2sds**, qui convertit notre format texte en un fichier ".sds". La section suivante détaille son fonctionnement.

B.2 L'application *txt2sds*

Cette application est basée sur le programme *SHOPP* et en reprend la routine de lecture de fichier. Elle demande donc le nom du fichier original, puis la manière dont sont modélisés les intervalles (milieu - demi-longueur, milieu - longueur ou borne inférieure puis supérieure). Elle demande ensuite s'il y a identification des individus, s'il y a le nom des variables, puis leur nombre, et enfin un nombre maximal d'individus (pour limiter la taille du tableau en mémoire vive). Une fois tous ces paramètres connus, notre programme lit le fichier de données et en stocke les valeurs, puis crée un fichier portant le même nom que le fichier d'origine (mais avec l'extension ".sds") et y écrit les entêtes nécessaires, puis la matrice de valeurs.

On obtient ainsi un fichier sodas qui nous permet de tester les méthodes de [SODAS, 2] sur nos jeux de données. Signalons que le jeu de données **cars** que nous avons analysé avec les méthodes de [SODAS, 2] dans la section 7.7 est le jeu normalisé, et non le jeu d'origine.

Remarquons que nous ne traitons avec *SHOPP* que des données intervalles (et éventuellement des données traditionnelles, si on a des intervalles de longueur nulle), et pas l'ensemble des données traitables par *SODAS*. Ainsi, notre application **txt2sds** ne peut créer de fichiers sodas que s'ils ne contiennent que des données de type intervalle.

Voici l'icone, puis l'affichage en console de l'application **txt2sds** :


```
 ##########################################################################
#                                                                          #
#                                                                          #
#    ##                ##        ########                      ##         #
# ######## ###    ## ########   ###    ###     ########  ########   ####### #
#  ###     ###   ###  ###            ####   ####       ###   ### ####      #
#  ###       #####    ###       #########     ########  ###   ### ########  #
#  ###    ####  ####  ###      ####                ####  ##    ###      #### #
#   ##### ###    ##    #####   ##########    #########   ####### #########  #
#                                                                          #
#                                                                          #
 ################################## TRANSFORM TXT FILES TO SODAS ONES #####

 Nom du fichier de donnees ? (sans l'extension '.txt')
 >

Comment sont présentés les intervalles ?
  0 : milieu - demi longueur
  1 : milieu - longueur
  2 : borne inf - borne sup
 >

 Y a-t-il identification des individus ? (1=OUI,0=NON)
 >

 Y a-t-il, en premiere ligne, le nom des variables ?
 (Un seul  nom pour les deux colonnes) (1=OUI,0=NON)
 >

 Nombre de variables, ou une variable est constituee d'une colonne Centre
 suivie d'une colonne Demi-Longueur ?    (s'il y a identification des
 individus, ne pas inclure cette variable dans le nombre des variables!)
 >

 Nombre maximum d'individus ?
 >
```